Nonparametric Optimization and Galactic Morphology

Brigham S. Anderson

CMU-RI-TR-03-17

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

August, 2003

Abstract

This thesis examines two areas of overlap between Memory-Based Reasoning (MBR) and stochastic optimization. The first area is a solution to a galactic morphology problem from astronomy: given a faint, distorted image of a galaxy, how can we determine its structure automatically and quickly? Solving this problem is of importance to the field of astronomy, since there are now hundreds of millions of such images which have not been analyzed due to computational costs. The new algorithm, GMorph, follows the MBR paradigm by populating image space with several million "model" galaxies ahead of time. During run-time, we just find the model image nearest to the unknown image. The more interesting issues are dealing with image distortion and speed. GMorph uses "eigengalaxies" to address both. Eigenspace has only a fraction of the number of dimensions that image space does, so deconvolution is faster, more stable, and more accurate. We achieve significant improvements in speed over previously published techniques.

We also introduce a new algorithm for optimization of similarity-based data. In a problem where only the similarity metric is defined, a gradient is rarely possible. The essence of MBR is the similarity metric among stored examples. The new algorithm, Pairwise Bisection, uses all pairs of stored examples to divide the space into many smaller spaces and uses a nonparametric statistic to decide on their promise. The nonparametric statistic is Kendall's tau, which is used to measure the probability that a given point is at an optimum. Because it is fundamentally nonparametric, the algorithm is also robust to non-Gaussian noise and outliers.

To my mother and father, to whom I owe just about everything

Acknowledgements

I would like to thank my advisor, Andrew Moore, who provided such an inspirational model throughout my graduate studies. Andrew was perpetually supportive and enthusiastic, and could be counted on to have a clever idea up each sleeve at any given time.

I am indebted also to my wife, K.C., for her heroic patience and love throughout the writing process. I also thank my adorable daughter, Aurora, who reminded me often and loudly what (or who) is really important in life.

Many thanks also to my thesis committee, Andrew Moore, Chris Atkeson (CMU RI), Leslie Kaelbling (MIT), Andrew Connolly (University of Pittsburgh, Astrophysics), and Matt Mason (CMU RI) for their ingenious questions and advice during the research process. Their suggestions for the thesis were all excellent and improved the document tremendously. I also thank Robert Nichol (CMU Astrophysics) for his ethusiastic discussions and support. David Cohen, currently at Google, was a creative early supporter of the work on pairwise optimization. Many enjoyable brainstorming sessions with Dave and Andrew were spent improving the algorithm.

Finally, I have to say that nothing whatsoever might have been done if not for the assistance of Suzanne Lyons-Muth in all matters administrative. Suzanne made the hoops of graduate school a pleasure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation: Memory Based Reasoning in Optimization

Memory-Based Learning[1] is a form of supervised inductive learning from examples. The algorithm is given labelled examples of different patterns that the learner might encounter during operation. After the learning phase, if a new pattern is presented to the algorithm, it will attempt to generalize from examples in its memory that are similar to the unknown pattern. The approach is based on a model of thought where reasoning occurs by direct reuse of stored experiences rather than the application of knowledge (such as rules or decision trees) abstracted from experience.

In memory-based learning, induction is based on the use of similarity [1, 53]. Similarity is a measure of how 'close' one pattern is to another. The main generalization technique employed by memory-based learning systems is the nearest-neighbor search. When using nearest-neighbor search to classify incoming patterns, a new query's similarity to all other instances in memory is determined, and the classification of the closest or most similar pattern in

[1] Also referred to as Lazy, Model-free, Nonparametric, Case-Based, Exemplar-Based, Similarity-Based, or Instance-Based learning.

memory is returned. There are numerous techniques for streamlining this basic strategy [7, 16, 34].

Memory-Based Reasoning has a long history in artificial intelligence among several disciplines (e.g., computer vision and language processing) [53]. Memory based reasoning can be viewed as a type of function approximator, and has been used in many domains, e.,g., control [4,33], locally weighted learning [3], natural language processing [10, 49], face recognition [51], and optimization [36].

The advantages of memory-based methods over other knowledge models include

- *transparency*, the learning process is easy to visualize and understand, and therefore is easy to work with.
- *flexibility in representation*, given enough examples, a memory-based learner can learn an arbitrary function [12].
- *fast learning*, learning an example is accomplished merely by storing the example in memory. No further processing is required.
- *incremental learning,* new examples can be learned at any time during the lifetime of the learner, and
- *locality,* the learner only uses data which is most similar/relevant to the query.

In this thesis we introduce two algorithms that perform nonparametric optimization. The first is a system that uses memory-based methods to determine the best-fitting morphology of two-dimensional images of galaxies. The second is an optimization technique that uses a nonparametric model of data during optimization.

Understanding galaxy morphology is a necessary step in understanding the formation of large scale structures in the universe. This is still an open area of research in astronomy; we currently do not know how certain galaxy shapes arise. A great deal of progress can be achieved by extracting data from large numbers of faint and noisy images, e.,g., whether the galaxy is spherical, elliptical, or disk-shaped, the size of the central bulge vs. the size of the disk, etc. The distribution of these structural parameters will aid in the testing of cosmological theories [52].

We introduce an algorithm addressing this galaxy morphology task. The algorithm must be able to fit a galaxy image model to a (possibly blurred) data image, and it must be able to do so quickly. After much trial and error, the most successful algorithm type was found to be Memory-Based Reasoning. In this case, memory-based means that an initial random population of synthetic images of galaxies are generated and then stored. When a query image comes in, it is compared to this population, and the nearest image's parameters are returned.

1.2 Overview of Thesis

The structure of the thesis will be the following:

Chapter 1 contains a brief overview of memory based reasoning and optimization.

Chapter 2 is a background of the major classes of nonlinear optimization techniques.

Chapter 3 describes the galaxy morphology problem

Chapter 4 proposes a new algorithm for solving it much faster than current techniques do. The algorithm uses MBR techniques in new domain.

Chapter 5 describes a new algorithm for optimizing in a memory-like space.

Chapter 6 discusses the conclusions of this work and directions for future research

Chapter 2

Optimization Background

Many disciplines have methods relevant to stochastic global optimization. Resistance to noise and experimental efficiency varies widely among these methods. Numerical methods have produced a large number of algorithms for locating optima. Generally, these algorithms have been designed to work on known functions with no noise. Their effectiveness within these domains and mathematical guarantees makes them popular examples of optimization algorithms.

Extremely efficient numerical algorithms exist to optimize one-dimensional functions without the use of gradients (e.g., Golden ratio, Brent's Method [45].) However, multidimensional problems have proven much more difficult. Numerical optimization of multidimensional functions mostly consist of line minimizations in combination with clever methods for determining search directions. Two well-known optimization algorithms that do not require gradients are conjugate directions and quasi-newton/variable-metric methods.

2.1 Optimizing Noisy Functions Without Gradients

Optimization of a process involves finding the highest peak (or deepest basin) on the underlying function surface. However, without prior knowledge of the shape of the function, this task requires an infinite number of experiments [56]. Therefore, we restrict our task to one of finding a good *local* optimum. We also restrict the task to minimization, since any maximization problem can be converted to one of minimization by changing the sign of the function.

One can express the problem of optimizing a function $f : \mathbb{R}^n \to \mathbb{R}$ as finding an input $\boldsymbol{x}$ to f that produces the smallest output $f(\boldsymbol{x})$. The strict definition of *global minimum*[1] $\boldsymbol{x}^*_{glob}$ for f is

$$f(\boldsymbol{x}^*_{glob}) < f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in V(\boldsymbol{x}), \boldsymbol{x} \neq \boldsymbol{x}^*_{glob}$$

where $V(\boldsymbol{x})$ is the set of feasible values of the variable $\boldsymbol{x}$. A point $\boldsymbol{x}^*_{loc}$ is a *strong local minimum* of f if

$$f(\boldsymbol{x}^*_{loc}) < f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in N(\boldsymbol{x}^*_{loc}, \eta), \boldsymbol{x} \neq \boldsymbol{x}^*_{loc}$$

where $N(\boldsymbol{x}^*_{loc}, \eta)$ is defined as the set of feasible points contained in the neighborhood of $\boldsymbol{x}^*_{loc}$, i.e., all points within some arbitrarily small distance η of $\boldsymbol{x}^*_{loc}$.

Alternatively, if the function f is smooth, for the vector $\boldsymbol{x}^*$ to be a local minimum of f the following is both necessary and sufficient:

$$\begin{aligned} &a.\ \|\nabla f(\boldsymbol{x}^*)\| = 0 \\ &b.\ \nabla\nabla f(\boldsymbol{x}^*)\,\text{is positive definite} \end{aligned} \tag{2.1}$$

However, often during optimization of some processes, one cannot measure f, ∇f, or $\nabla\nabla f$, but only the output y. The observed output y is the sum of $f(\boldsymbol{x})$ and some random variable ε. This source of noise, ε, we assume to be

[1] Excludes the possibility of multiple global optima.

independent, identically distributed[2], and have a mean of zero.

$$y(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon \tag{2.2}$$

Noise contamination, ε, confounds the optimization problem. Noise can mask the presence of true optima and create illusions of optima where none exist. Noise can be created by measurement errors, variations in the environment, and inherent randomness in the process. The type of noise is another factor, methods will have differing success with Gaussian noise, non-Gaussian noise, and outliers.

The actual underlying function, f, of empirical processes is usually unknown. Non-differentiability results from this, since the experimenter cannot analytically determine gradients. The lack of gradient is a significant factor in the choice of algorithm, since most algorithms use first or second derivatives, e.g., Steepest descent, Newton's method, Conjugate gradients, and Variable metric methods.

When optimizing an empirical process, the cost of experiments is more often a dominating factor than computational cost. Researchers and engineers in the real world have limited time and funding for exhaustive experimentation. Experiments with a chemical plant may cost thousands of dollars per experiment. A robot janitor may get the opportunity to test a parameter setting once a day. Because of cost, the number of function evaluations required can make or break algorithms optimizing empirical processes. In empirical processes, it is often the case that the cost of the experiments (or function evaluations) exceeds the cost of computation.

An example characterizing this type of problem is a biochemist who must set the temperature and enzyme concentration of a reaction in order to maximize yield. The underlying function $f(x)$ is the expected yield of the reaction. He has no idea what the underlying function or its gradients look like. The researcher can adjust the parameter vector x (temperature and enzyme concentration.)

[2]All the errors are independently generated from the same probability distribution.

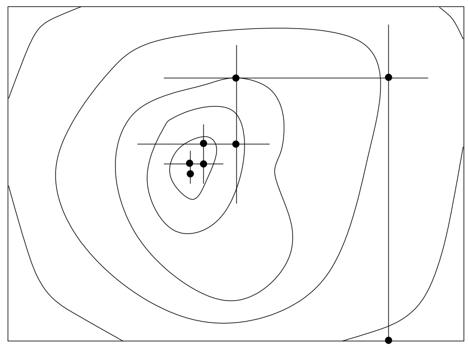


Figure 2.1: Coordinate Directions search. The search proceeds by performing sequential one-dimensional searches (represented by line segments) along alternating coordinates.

The output $y(x)$ represents the observed yield, and is corrupted by noise such as reagent variation, measurement error, and technician error. The task of the optimizer is to find x^*, which is the combination of temperature and enzyme concentration with the greatest expected yield. The cost of each dosage of enzyme is much more than the cost of the computer time per experiment.

2.2 Systematic Search

2.2.1 Coordinate Directions

Coordinate descent forms the basis for directional descent algorithms. These algorithms start with some set of search directions $\mathcal{P}$, and a one dimensional search is conducted along each of these directions in turn. After each search, the current best-guess location for the optimum is updated to be the best location found along the search direction. Coordinate descent is the simplest type of directional descent: the search directions are the standard basis vectors, $\mathcal{P} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_d\}$. Hence only one variable at a time is modified. Figure 2.1 shows an example of this process. In most cases this is an inefficient use of function evaluations because coordinate directions does not try to 'learn' anything about the local gradient during the search.

2.2.2 Conjugate Directions/Powell's Method

Rather than specifying the search vectors a priori, the standard basis directions are modified as the algorithm progresses. Conjugate directions methods are the gradientless siblings of conjugate gradient descent. These algorithms begin at a start point and develop a set of search directions which are all mutually conjugate w.r.t. the estimated Hessian at that point (i.e., for search directions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, and Hessian A, it is true that $\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0$ for all $i \neq j$.) Assuming that the function is quadratic and positive definite around the minimum, performing line searches along these directions will minimize the function in d steps in a d-dimensional space. The trick, of course, lies in selecting a search direction set that satisfies these conditions. There are multiple approaches to this, one of which is Powell's method [44,45].

Powell's method ensures that, when the procedure is applied to a convex quadratic function, d mutually conjugate directions are generated after d sweeps. Powell's method proceeds as follows:

- Set $\mathcal{P}_0$ equal to the standard basis vectors $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_d\}$
- Set $\boldsymbol{x}_0$ as the starting position for the search
- Point $\boldsymbol{x}_{k+1}$ is obtained by starting at $\boldsymbol{x}_k$ and moving to the minimum along each search direction in $\mathcal{P}_k$ (sequentially, and in the order oldest to newest.)
- P_{k+1} is the set P_k with the vector $(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$ added and the oldest direction removed.

The problem with Powell's method is that the directions will generally all be linearly dependent to some degree after d iterations. They will become increasingly so as the search progresses, and hinder or halt exploration. There are several techniques for preventing this, e.g., reinitializing the direction set, constructing a new set from the estimated Hessian, or selectively discarding directions that are linearly dependent.

2.2.3 Finite-Difference Approximations (FDA)

Another approach to the problem of optimizing gradient-free functions is to use some extra function evaluations to approximate the gradient. Such methods are generally known as finite difference approximations. One such approximation is the following

$$\nabla_i f(\boldsymbol{x}) \approx \frac{1}{h_i}\left[f(\boldsymbol{x} + h_i \boldsymbol{e}_i) - f(\boldsymbol{x})\right]$$

for suitably chosen (i.e., small) intervals h_i, where $\boldsymbol{e}$ are the basis directions. These approximate gradients can then be used in a gradient-based method such as steepest descent, quasi-Newton descent, or conjugate gradient descent. These methods will often provide more than enough savings computationally to justify the use of finite difference approximations.

2.2.4 Quasi-Newton Descent with FDA

Quasi-Newton methods are also known as Variable Metric methods [14]. Although as originally designed they need gradient information during operation, they can use finite differences to create an approximation to the gradient.

Quasi-Newton methods use function gradients obtained during search to continuously build up an approximation to the inverse Hessian (which is forced to be symmetric positive definite.) It then uses this estimate to take a step in the 'Newton direction' ($\boldsymbol{u} = -\widehat{H}^{-1}\nabla f$) [15]. Under many conditions, Quasi-Newton descent offers excellent convergence to an optimum. However, it can oscillate or diverge to infinity under some, and is susceptible to noise.

2.2.5 Nelder-Mead Simplex Search

The Simplex method [39] (no relation to linear programming), is an intuitive and aesthetic optimization algorithm that does not use gradients. Simplex search optimizes over a k dimensional space using a simplex (i.e. a $k+1$ dimensional polyhedron). The algorithm evaluates the function at each vertex, then reflects

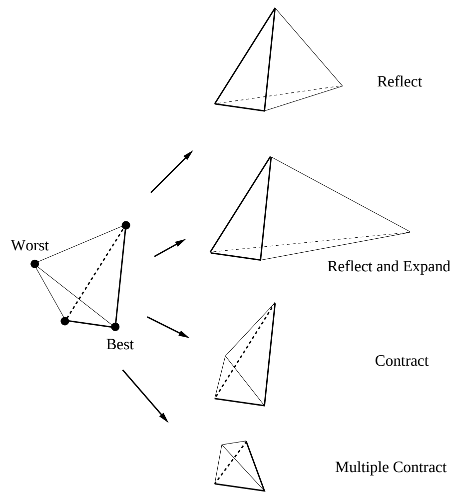


Figure 2.2: Simplex algorithm. At each step, the algorithm changes the polyhedron's shape to adapt to the terrain and move toward the optimum. The different movement options depend on the environment, e.g., the default step is reflection, but if the trial step is better than the best so far, then reflect and expand.

the worst-performing point through the body of the simplex. This operation produces a new simplex which has moved up the estimated gradient. Basic simplex transformations let the simplex shrink near the optimum, grow in large linear zones, and ooze along ridges.

The simplicity of Nelder-Mead makes it an appealing and easy to implement method for optimization. If experiments are cheap, then Nelder-Mead can be an appropriate optimization method. However, Nelder-Mead does not use its experiments efficiently, since it only keeps the most recent $k+1$ results and uses a heuristic for the selection of points. In the absence of noise, it is inferior in this respect to conjugate directions methods such as Powell's method [45].

2.2.6 Experiment design and Response surface methods

Response Surface Methodology (RSM) [6, 43] was first developed in chemical engineering to optimize chemical manufacturing processes. Since then, it has been used extensively in many manufacturing fields. The basic building blocks

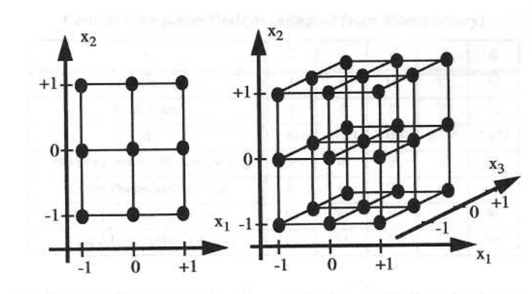


Figure 2.3: Three-level Factorial design for a 2- and 3-dimensional input space.

of RSM are experimental designs. An experimental design is a set of points in search space in a pattern that will produce good data points for a linear or polynomial regression. Examples of such designs are factorial designs and central composite designs, which usually have special properties that make them experimentally efficient [5]. Two common designs are in Figures 2.3 and 2.4.

RSM's approach to optimization has the researcher first identify a Region of Interest (ROI) from which the experiments are to be taken. An experimental design is generated and the resulting data points are used to fit a low-order polynomial. Points continue to be taken until the regression indicates a significant fit. Once the regression estimates the gradient confidently, the ROI moves down the gradient, and the process repeats. Quadratic regression locates optima within the ROI. RSM traditionally requires the judgment of a statistician during most of the experimental procedure, since it is a methodology and not an algorithm.

Since RSM was designed for optimization of real-world processes such as chemical reactions, it was made to be robust to Gaussian noise. RSM avoids changing operating conditions based on inadequate evidence, so only moves once the data justifies it statistically. So, experimental efficiency is generally good,

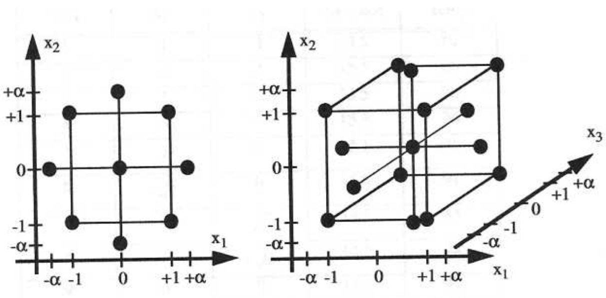


Figure 2.4: Central Composite design for a 2- and 3-dimensional input space.

but the experimental designs require many sample points in even moderate dimensions. The most binding constraint of RSM, however, is its need for human judgment at most of its stages, for instance, choosing the size and shape of the ROI for the bias-variance tradeoff and choosing the experimental design.

2.2.7 PMAX and IEMAX

Based on the data from previous experiments, PMAX uses a non-linear function approximator to estimate the underlying function f. PMAX then takes the next experiment at the optimum of the *estimate* of $f(x)$. This approach has been used with a decision-tree approximator [50], with neural nets (in many commercial products), and with locally weighted regression [36]. Variations of PMAX include taking the next experiment not at the predicted optimum, but instead where the confidence intervals are widest, or where the top of the confidence interval is maximized [36], or in accordance with the Interval Estimation heuristic [21] (see Figure 2.5.) The algorithm using the interval estimation heuristic we we call IEMAX [36]. IEMAX has a principled tradeoff between exploring unknown space and exploiting knowledge of better-known space to improve the best-so-far solution.

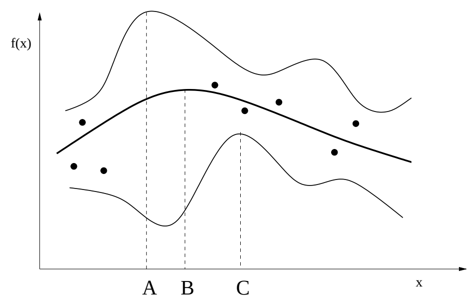


Figure 2.5: PMAX and IEMAX algorithms in action. The middle line is the best-fitting model surface, while the upper and lower lines are 95% and 5% confidence interval boundaries, respectively. PMAX would choose B for the next experiment, a risk-seeking IEMAX algorithm would choose A, and a risk-averse IEMAX would choose C.

If a section of confidence interval has a high upper bound, this suggests that either that area of space has little information, or that there is information that the underlying function is good. High upper bounds can also indicate the presence of noise or an inability of the function model to fit the underlying function (bias).

The tradeoff between exploration and exploitation is determined by the confidence interval size. A confidence interval of 95% will be exploratory, and an interval of 10%, would be risk-averse. For example, asking the algorithm to maximize the 1% bound would be roughly equivalent to trying to find the best of all worst-case scenarios.

IEMAX provides an intuitive approach to optimization that is very flexible in the type of model to use for the underlying function and in the experiment selection criteria. The models described above can all handle noise, so IEMAX can be robust to noisy functions. PMAX and IEMAX can also use all the data gathered for each decision, so they can theoretically have high experimental efficiency.

Empirically, it has been found that PMAX using locally weighted regression as the function approximator is often faster than more sophisticated alternatives [36]. However it has some serious drawbacks: First, one must solve the bias-variance tradeoff, which is roughly how smooth to make the function. This is often determined automatically using cross-validation [35], but this proves difficult with the set of very few, weirdly distributed data points which result from optimization runs. Empirically, very poor performance has been observed when attempting this. Second, PMAX can be expensive; it needs to train a function approximator each time an experiment is made and that function itself must be numerically optimized to produce the suggested experiment. Third, in the presence of noise, PMAX can get stuck in "hallucinated" optima, as illustrated in Figure 2.6.

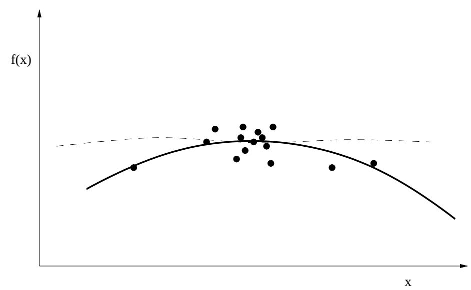


Figure 2.6: PMAX in some difficulty. The dashed line is the true function, and the solid line is PMAX's current model of the function. Some unlucky early points have created a spurious optimum, which PMAX will not escape from.

2.3 Random Search

2.3.1 Stochastic Approximation

All stochastic approximation algorithms are closely related to gradient descent, they just have approximate gradients and clever ways of decreasing step size. The standard algorithm of stochastic approximation (SA) for this problem has the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \hat{g}_k(\boldsymbol{x}_k)$$

where $\hat{g}_k(\boldsymbol{x})$ is an approximation of the gradient ∇f at $\boldsymbol{x}$, and $\{\alpha_k\}$ is a sequence of positive scalars that decreases to zero while satisfying certain special properties [26].

The original Robbins-Monro algorithm [48] is a stochastic approximation method that functions by placing conditions on step sizes and whose convergence is guaranteed under mild conditions. However, the original method requires knowledge of the analytical gradient of the function.

Keifer-Wolfowitz (KW) [23] is a finite difference version of the Robbins-Monro algorithm that maintains the nice convergence properties, while eliminating the need for an analytical gradient. This algorithm estimates the gradient at each step by sampling points in both directions along each dimension of the input space. Based on the estimate, it moves its experiment center by a certain step size and repeats. The step size decreases slowly in such a way as to ensure convergence.

KW's two main advantages are its simplicity and convergence guarantees. In addition, its behavior in the lab suggests that stochastic approximation will aggressively explore its environment [36]. However, noise can cause it to attempt wild experiments. The total number of experiments can also become large, since it throws away all data after each gradient estimation and descent step.

2.3.2 Simulated Annealing

Similar in form to stochastic approximation, the Simulated Annealing method was created with an analogy to the physical process of an annealing metal in mind. It therefore requires a temperature parameter T, and a temperature schedule for reducing T gradually over time. Simulated annealing's major advantage over other methods is an ability to avoid becoming trapped at local minima. The algorithm employs a random search which not only accepts small steps that better the objective function f, but also some that worsen it. The latter are accepted with a probability

$$p \propto exp\left(\frac{\Delta f}{kT}\right) \tag{2.3}$$

where Δf is the change in f and k is Boltzmann's constant. This basic method is called the Metropolis algorithm [30]. Because this is basically the controlled introduction of noise, Simulated Annealing is robust to noise inherent in the target function.

For problems with continuous input values, [57] recommends that new trial solutions be generated according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + Q\boldsymbol{u} \tag{2.4}$$

where $\boldsymbol{u}$ is a vector of Gaussian random numbers, each having zero mean and unit variance, and Q is a matrix that controls the step size distribution. The matrix Q is found by solving

$$S = QQ^T \tag{2.5}$$

where S is an estimate of the covariance of the local topography. The equation can be solved by Cholesky decomposition, for example. The end result of Equations 2.4 and 2.5 is to generate random steps having covariance S. The S is updated as the search progresses to include information about the changing local topography:

$$S_{i+1} = (1-\alpha)S_i + \alpha\omega X$$

where matrix X measures the empirical covariance of the path actually followed and the damping constant α controls the rate at which information from X is folded into S with weighting ω. One drawback of this scheme is that the solution of Equation 2.5, which must be done every time S is updated, can represent a substantial computational overhead for problems with high dimensionality. In addition, because the probability of an objective function step being accepted, given by Equation **??**, does not reflect the size of the step taken, S must be estimated anew every time the system temperature is changed.

2.3.3 Evolutionary computation

Evolutionary methods all attempt to maximize a fitness function, so can also be used as general-purpose optimization routines. The basic evolutionary model maintains a population of experiments and preferentially "breeds" the parameters of experiments with good outcomes. The population will moves towards an optimum over time [8, 17].

One example of an evolutionary scheme for optimizing in continuous space is Differential Evolution [54]. Optimization by this method is similar to genetic algorithm optimization, except that the creation of new members of the population is achieved by the following: take two random points from the population and find their difference vector by subtracting the worse point from the better point. This difference vector is then multiplied by a user-specified weight and added to a third randomly chosen point, thus producing a new candidate. Optionally, the third random point could have been a combination between the best point of that generation and itself.

Evolutionary methods are attractive because they can perform a wide variety of optimization tasks such as noisy, discontinuous, and high-dimensional problems. However, the properties of this type of optimization are understood less well than most other techniques, and their superiority over hillclimbing has not been established [31]. In addition, these approaches tend to converge slowly; needing thousands, sometimes millions, of evaluations. Thus the total experimental cost of evolutionary methods can be prohibitively expensive.

2.3.4 Q2

The Q2 algorithm [37] attempts to achieve second order (quadratic) convergence by fitting local quadratics in a Newton-like method. Unlike RSM it attempts to automatically determine a good region of interest, i.e. a region predicted with high probability to contain a local optimum that is as small as possible (so as to minimize bias in the quadratic approximation) but not so small that there is no confidence in the optimum location. A simplified description of Q2's thinking for each experiment is to first fit a quadratic to all the data, then eliminate the point that the quadratic would predict to be the worst (*not* the actual worst point or the worst-fitting point.) That point is eliminated and the process repeated until only a few points remain. These points are assumed to be living in a promising part of the input space, so the experiment will be drawn close by.

One of Q2's advantages is that it uses all of the data to choose each experiment. The quadratic regression also allows the algorithm to handle Gaussian noise in a statistically justified manner. Q2 thus performs well on a variety of noisy tasks subject to the assumptions of a continuous underlying f and Gaussian noise. Experimental efficiency appears competitive with Nelder-Mead and PMAX.

Q2 assumes Gaussian noise in the function, so outliers can throw off the quadratic regression badly. The algorithm also assumes a quadratic underlying function, so it can experience bias during function approximation. The magnitude of this bias tends to peak in the beginning stages of the algorithm, when Q2 attempts to fit a single quadratic function to large chunks of the dataset.

2.4 Least-Squares Model-Fitting

Specific optimization methods exist for least-squares model-fitting problems, e.,g., Levenberg-Marquardt [29] and the Gauss-Newton method [15]. These are similar to other optimization techniques, except that they take advantage of a differentiable model function and the structure of the target function:

$$f(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{n}(\psi_i(\boldsymbol{x}) - \boldsymbol{y}_i)^2$$

or

$$f(\boldsymbol{x}) = \frac{1}{2}\left\|\psi(\boldsymbol{x}) - \boldsymbol{y}\right\|_2^2$$

where $\psi(\boldsymbol{x})$ is the model function and $\boldsymbol{y}$ is the data to be fit. In this case the target function $f(\boldsymbol{x})$ is the sum of squared errors between the data and the model. If we can compute the Jacobian $J(\boldsymbol{x})$ of the model function, then we can use this information to speed up search.

In the case of Levenberg-Marquardt, noise is assumed to be Gaussian, so the error function is least squares. Given the Jacobian $J(\boldsymbol{x})$ of the model function, the gradient and Hessian of the error function f have a special structure [15]:

$$\begin{aligned} g(\boldsymbol{x}) &= J(\boldsymbol{x})^T f(\boldsymbol{x}) \\ G(\boldsymbol{x}) &= J(\boldsymbol{x})^T J(\boldsymbol{x}) + Q(\boldsymbol{x}) \end{aligned} \tag{2.6}$$

where $Q(\boldsymbol{x}) = \sum_{i=1}^{n} f_i(\boldsymbol{x}) G_i(\boldsymbol{x})$. Equation 2.6 shows that the Hessian is a combination of first and second-order terms. Most least-squares methods rely on the assumption that the first-order terms will come to dominate near the optimum. If, however, the residuals are large near the optimum, then there is little advantage to using a least squares method. Large residuals can be characterized by their effect on the terms of Equation 2.6, e.g., the relative sizes of $\|f(\boldsymbol{x}^*)\|$ and the largest eigenvalue of $J(\boldsymbol{x}^*)^T J(\boldsymbol{x}^*)$.

2.5 Summary

We have covered a many of the methods used to optimize a function in the presence of noise. One manner in which to organize these methods is by the feature of stochasticity, i.e., whether the algorithm is deterministic or not. Deterministic algorithms tend to assume a specific local model of the data, generally quadratic, as in the case of quasi-Newton descent and conjugate directions. Stochastic algorithms tend to be less committed to (or more suspicious of) their model and more willing to accept points that contradict it. Because of this, although not a hard rule, stochastic algorithms are generally better able to handle noisy environments.

Chapter 3

Galaxy Morphology Background

This chapter describes an image model-fitting problem from astronomy, as well as a new method, GMorph, for solving it. The novelty of GMorph is its use of a memory-based search and linear regression in a reduced-dimension eigenspace. GMorph is over two orders of magnitude faster than the existing competition, and is as accurate.

3.1 Problem

In order to understand the formation of large scale structures in the universe, a necessary step is the understanding of galaxy morphology. This is still an open area of research in astronomy; we currently do not know how certain galaxy shapes arise. The distribution of shapes and their correlation with other measured properties of galaxies is important for generating and testing hypotheses about galaxy morphology. This requires extracting various types of information from large numbers of faint and noisy images, e.,g., whether the galaxy is spherical, elliptical, or disk-shaped, the size of the central bulge vs. the size of the

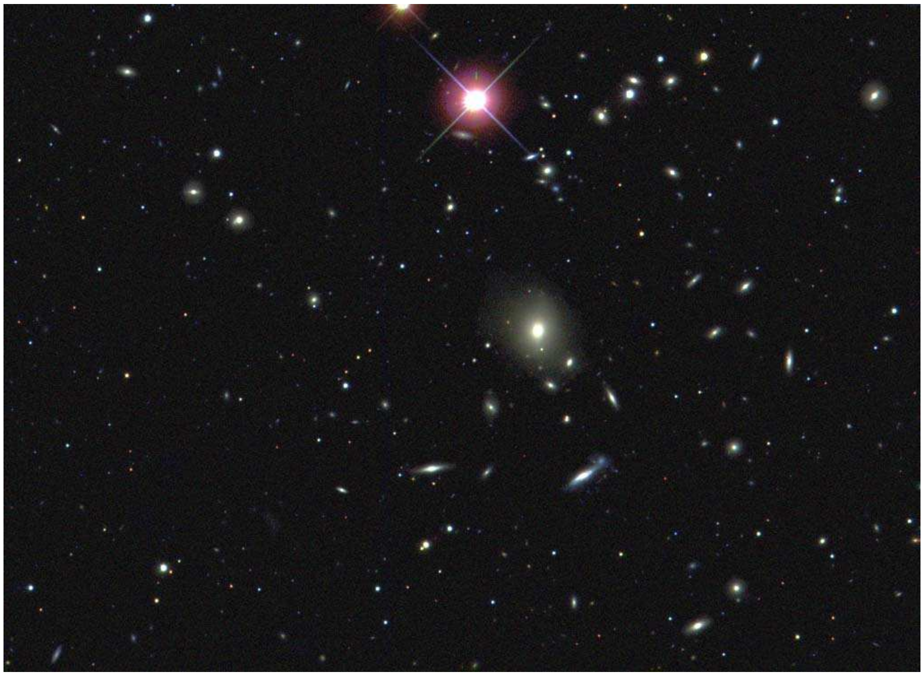


Figure 3.1: Image from Sloan Digital Sky Survey (SDSS). Most of the elliptical objects are galaxies.

disk, etc. In order to test cosmological theories which are under-constrained, we would like to know the distribution of these structural parameters [52].Some examples of these types of images are in Figure 3.3.

A common classification system still in use today is the Hubble tuning fork (Figure 3.2), which roughly divides all galaxies into ellipticals (bulge-dominated) and spirals (disk-dominated) [19]. The corresponding structural parameter is the bulge-to-disk ratio, which describes the relative amounts of light emanating from the two components. Hence a coarse galaxy classification of can be made directly from a single structural parameter, which can be obtained by fitting a mathematical model to a galaxy image.

There are significant obstacles, however, to fitting these models. Taking photographs of galaxies from ground-based telescopes involves looking up through the atmosphere and having light blurred by the different diffraction indices

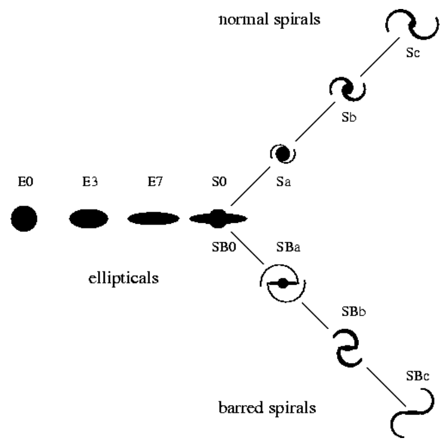


Figure 3.2: Hubble tuning fork diagram. The fundamental division of galaxies is into ellipticals (E) and spirals (S). The number after the ellipticals is the ratio of their major axis to their minor axis, called the ellipticity. The size of the central bulge diminishes from left to right.

caused by temperature differences in the air layers. Distortion also results from lens imperfections (telescope, filters, mirrors, lenses, etc.) However, the atmospheric effects are more troublesome because they vary most quickly over time. Figure 3.7 shows examples of these effects.

All these distortions are summarized in the Point Spread Function (PSF), which in this case takes the form of a single image. As its name implies, the point spread function is the spread that a single point of light would have undergone had it traveled the same path as the light from the original image. Often the PSF is obtained from the appearance of a nearby star, which serves as a point source.

3.2 Notation

Unless noted otherwise, all images are assumed to be matrices, and their pixels will correspond to elements in the matrices.

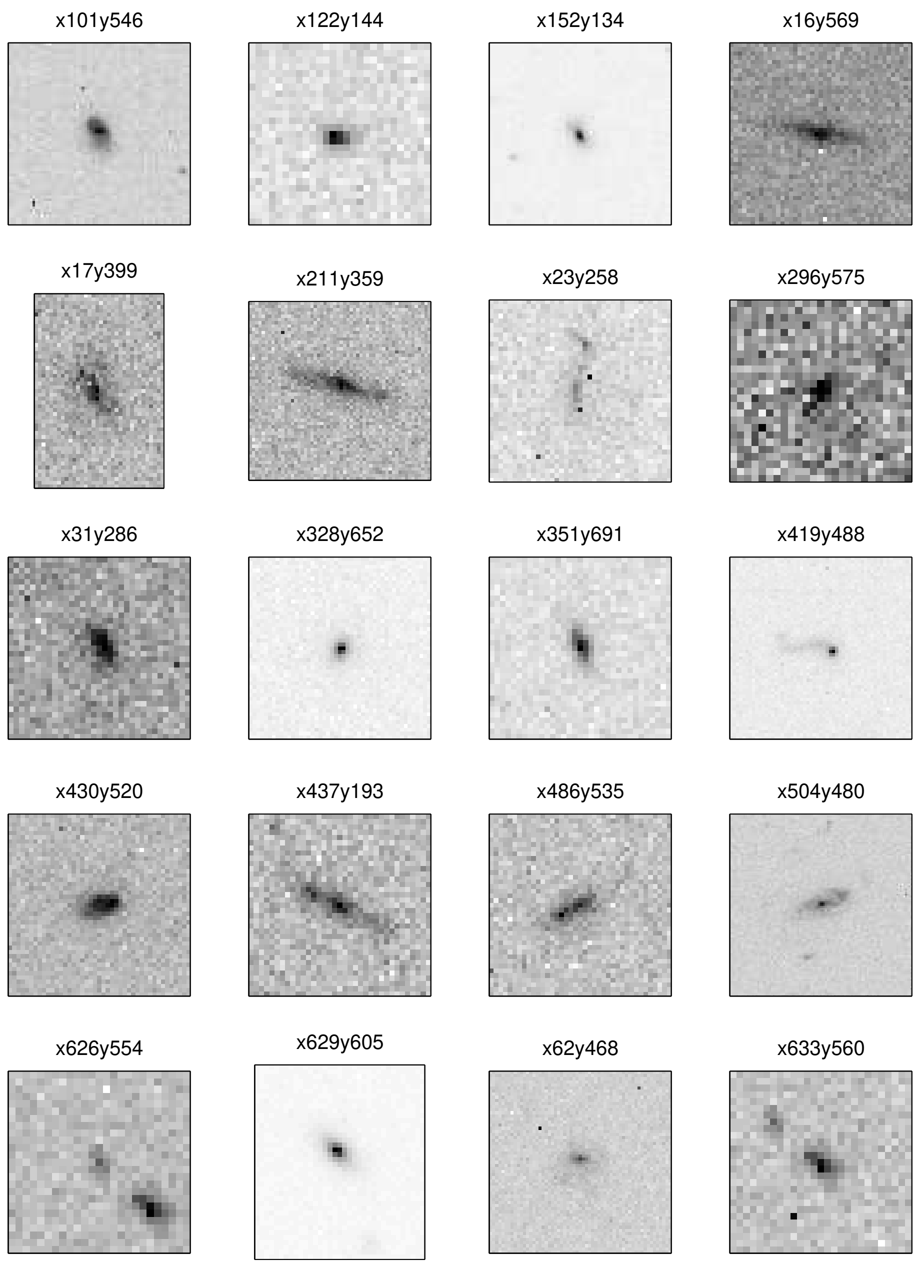


Figure 3.3: Galaxy images taken from the Groth Survey Strip, filter I_{814}, Field 8, wide field camera chip 3. Image scale $0''.1/pixel$ and total exposure time 4400 seconds [52].

All matrices will be capital letters (X), all vectors will be lower-case boldfaced letters ($\boldsymbol{x}$). Elements of matrices will be denoted by the lower-case letter with two subscripts indicating the row and column of the element (x_{ij}).

Often, it is more convenient to deal with images as vectors (e.g., when referring to them as points in Euclidean space.) The lower-case boldfaced version of the matrix's letter will correspond to its vector form. In this case, we assume that the columns of the matrix have all been vertically concatenated. If we wish to make the vector conversion explicit, the $\overrightarrow{\bullet}$ operator will be used.

For example, the $n \times n$ image X can also be represented as a vector, $\boldsymbol{x}$ or $\overrightarrow{X}$, which has dimensions $n^2 \times 1$. The ith column of X is $\boldsymbol{x}_i$ and the element in X's jth row and kth column is x_{jk}.

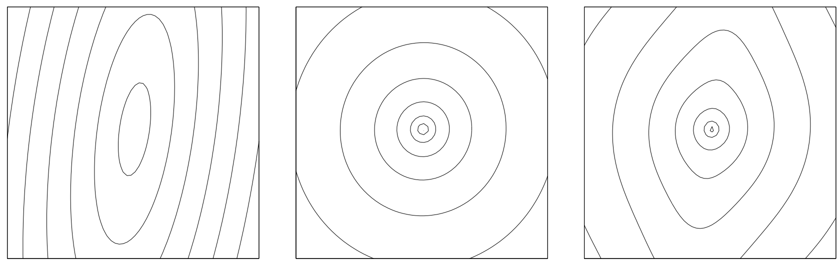
3.3 Modeling Galaxy Images

Here we use a model that generates an image from a vector of structural parameters. We will sometimes refer to this as a forward model to differentiate it from its inverse, which is ultimately the goal. The model $\mathcal{M}$ is a tuple of three elements: $\{\Theta, \Psi, S\}$, where

- Θ is the space of all possible surface brightness parameters,
- Ψ or $\boldsymbol{\psi}$ is the surface brightness function, and
- S is a matrix, the PSF, which describes the atmospheric and other distortions.

We assume that the parameter vectors from Θ have a uniform distribution, so $\mathcal{M}$ can be interpreted as a generative model of how galaxies are produced. The inner workings of $\mathcal{M}$ are summarized in the following sequence of transformations:

$$\Theta \xrightarrow{\Psi} R^N \xrightarrow{S\star} R^N$$



(a) pure disk (b) pure bulge (c) 0.5*disk + 0.5*bulge

Figure 3.4: Contours of bulge and disk images. The contour lines are at powers of 10 and are at equivalent intervals between images. The disk is tilted 70° with an 18 pixel radius, and the bulge has 0 ellipticity with a 25 pixel radius. The images are 64×64.

A vector of parameters from Θ is fed into the surface brightness function Ψ, which produces an image with N pixels (in the $\mathbb{R}^N$ space). This image is then convolved with S, the PSF image, to create an output image in $\mathbb{R}^N$. Perhaps counter-intuitively, convolution is a linear operation, since each resulting pixel is a linear combination of all pixels. Convolution with S could be imagined as multiplying an image vector by a $N \times N$ PSF matrix.

As mentioned, the model image $\boldsymbol{\psi}(\boldsymbol{\theta})$ is just a linear combination of two component image functions,

$$\boldsymbol{\psi}(\boldsymbol{\theta}) = \boldsymbol{\psi}_{disk}(\boldsymbol{\theta}) + \boldsymbol{\psi}_{bulge}(\boldsymbol{\theta})$$

which we will refer to as the disk image $\boldsymbol{\psi}_{disk}(\boldsymbol{\theta})$, and the bulge image $\boldsymbol{\psi}_{bulge}(\boldsymbol{\theta})$. Empirically, galaxy luminosity profiles show that surface brightness of galaxies fall into two broad categories: exponential (the disk) and de Vaucouleurs profiles (the bulge) [11].

Disk Component, Ψ_{disk}

A 2-dimensional, infinitely large disk approximates the galaxy's spiral component. The disk is bright in the center, and moving outward fades into transparency. No other structure, such as spiral arms, are modeled. This necessary simplification presents little difficulty in practice because the far-away galaxies of interest tend to be faint, relatively small with respect to the PSF, and/or of low resolution. Any spiral structure also tends to be visible in the residuals of the fit.

The surface brightness $\psi_{disk}(r)$ of the disk of a spiral galaxy has empirically been found to have an exponential form [13, 25]. A common model consists of an infinitely thin disk with brightness tapering off exponentially away from the center. In the disk, the intensity as a function of the distance from galactic center has an exponential profile of the form

$$\psi_{disk}(r) = \quad \psi_0 exp(-\tfrac{r}{r_d}) \tag{3.1}$$

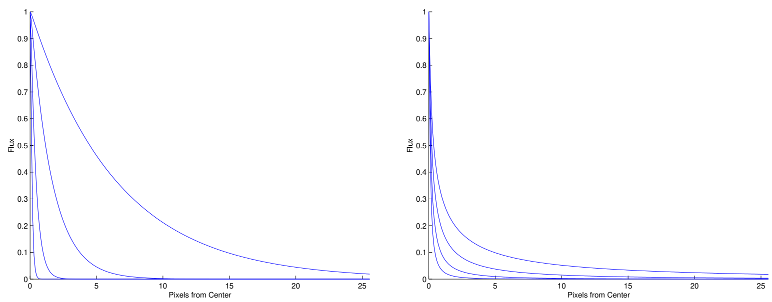
where ψ_{disk} is brightness, ψ_0 is the central brightness of the disk, and r_d is the disk radius. This profile is illustrated in Figure 3.5. The total light of the disk is given by

$$F_d = \int_0^{\infty} \psi_{disk}(r) 2\pi r \, dr = 2\pi r_d^2 \psi_{disk}(r_d)$$

Bulge Component, Ψ_{bulge}

The bulge will be modeled with a classical de Vaucouleurs profile. Also known as the $r^{1/4}$ law, de Vaucouleurs' law is perhaps the most widely used empirical law to describe the surface brightness profile of an elliptical galaxy (see the Hubble 'tuning fork' in Figure 3.2.) Assuming a perfectly circular bulge, the bulge intensity relative to the distance from the center of the galaxy has the form

$$\psi_{bulge}(r) = \psi_e exp\left(-b\left[\left(\frac{r}{r_e}\right)^s - 1\right]\right) \tag{3.2}$$



(a) Disk profile (b) Bulge profile

Figure 3.5: Disk and bulge profile examples. Moving from left to right inside the figure, the profiles have 1,4,16, and 64 pixel radii. The total flux has been manipulated for all 8 profiles so as to have the brightness at the center equal one. Otherwise, the larger radii profiles would be invisibly flat.

where ψ_e is the brightness of the bulge at the radius r_e. The constant s is set to $\frac{1}{4}$ in the de Vaucouleurs profile, and will be this value during all experiments in this thesis. The constant b is equal to $(1.9992 * s - 0.3271)$, and has been set such that the scale length, r_e, would contain half of the total light of the galaxy in the 2-dimensional image. The parameter r_e is therefore also referred to as the half-light radius. The total light F_b of the bulge is given by

$$F_b = \int_0^{\infty} \psi_b(r) 2\pi r \, dr = 2 \int_0^{r_e} \psi_b(r) 2\pi r \, dr = 7.22\pi r_e^2 \psi_e$$

See Figures 3.5 and 3.4 for examples of these profiles.

Projecting a Galaxy onto an Image

The bulge and disk models as described so far are functions only of radius; they are perfectly circular with one parameter each to describe the 'radius' of the brightness function. The disk is a two-dimensional object suspended in three-dimensional space, and projected onto a two-dimensional plane to produce its image. The bulge just describes a brightness function on the image; it is not interpreted as a three-dimensional structure due to its inherent ambiguity. For

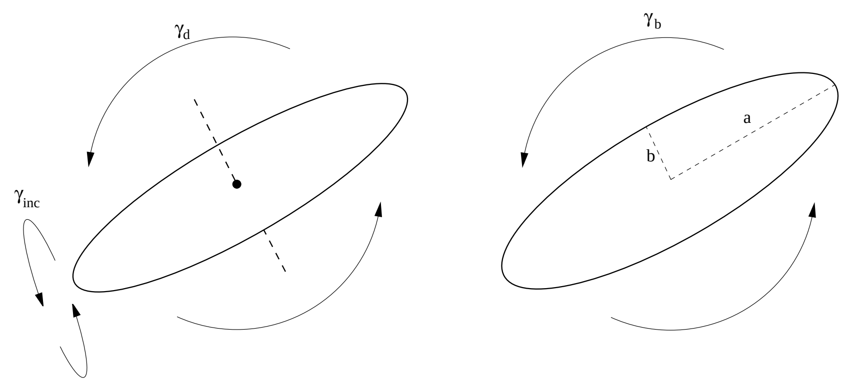


Figure 3.6: Disk and bulge. The ellipses mark all points of equal distance, r, from the center. The disk component, left, is interpreted as a 2-dimensional disk in 3-dimensional space, while the bulge is a 2-dimensional ellipsoid.

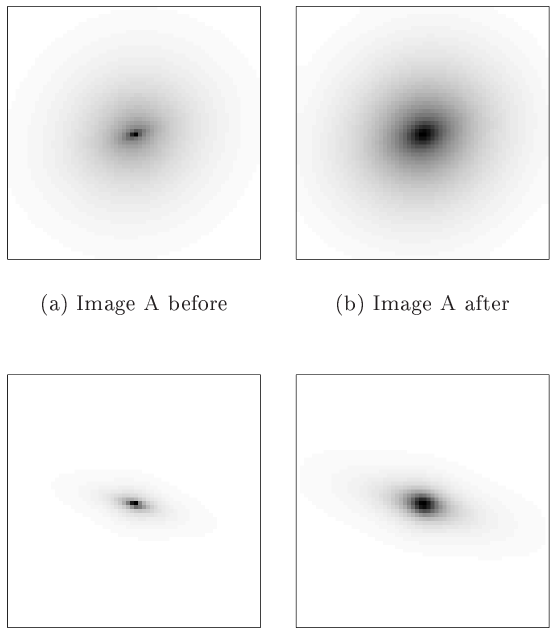
example, there is no way to tell the difference between a spherical bulge and a cigar-shaped bulge pointing at the viewer.

For both bulge and disk, their shape is modified by two other parameters each: a squashing parameter and a rotation parameter. The projection onto the x-y plane takes the following form:

$$A = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \tag{3.3}$$

$$r = \sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^T (A^T A)(\boldsymbol{x} - \boldsymbol{\mu})} \tag{3.4}$$

where A describes the squashing and rotation of the profiles, $\boldsymbol{\mu}$ is the x,y position of the center of the galaxy, and r is the radius in either Equation 3.2 or Equation 3.1. In the case of the disk, the squashing results a straightforward physical process: the galaxy can be inclined toward the viewer. The parameter is γ_{inc} and is expressed in degrees. Figure 3.6 illustrates the difference between inclination and rotation here. For the disk, the a in Equation 3.3 is 1, and the b is $\frac{1}{\cos^2 \gamma_{inc}}$. This means that the disk is first rotated about the x axis, then rotated about the z axis.



(a) Image A before (b) Image A after

(c) Image B before (d) Image B after

Figure 3.7: Two synthetic galaxies (A and B) before and after convolution with a Gaussian PSF with a standard deviation of 2 pixels. The images are 64×64 pixels.

The bulge's compression is described by its apparent ellipticity, ϵ. Ellipticity describes the relationship between an ellipse's semi-major axis a and its semi-minor axis b and is defined to be

$$\epsilon \equiv 1 - \frac{b}{a} \tag{3.5}$$

For the bulge, the b in Equation 3.3 is identical to the given constant in Equation 3.2, and is fixed at 7.6697. The intrinsic ellipticity ϵ' is related to the apparent ellipticity by $\epsilon = \epsilon' \cdot \sin i_b$, where i_b is the inclination of the bulge. The semi-major axis a is then determined by $\frac{b}{1-\epsilon}$.

The Point Spread Function, S

The bulge and disk images are added together, then convolved with a Point Spread Function (PSF). As mentioned, the PSF image describes the distortions incurred by the image from sources such as lens imperfections and atmosphere. We assume here for simplicity that the PSF image is the same size as the images produced by Ψ, $n \times n$. Due to the physical interpretation of S, it must satisfy the following constraints:

$$s_{ij} \geq 0 \quad for\, i = 1, 2, \ldots, n\, j = 1, 2, \ldots, n \tag{3.6}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} s_{ij} = 1 \tag{3.7}$$

Applying the PSF to an image involves convolving the two images. The formula for convolving discrete, two-dimensional functions which are only nonzero on an interval $1 \leq i \leq n$, $1 \leq j \leq n$ is

$$(S \star \Psi)_{ij} = \sum_{k=1}^{n} \sum_{l=1}^{n} s_{i-k,j-l} \psi_{kl} \tag{3.8}$$

If S and Ψ are images, then the zero padding would then be trimmed from the result. The awkward edge effects of dealing with images are handled here by generating the synthetic image larger than normal, convolving, and trimming the excess edges.

This step is considerably accelerated by taking advantage of the Fast Fourier Transform (FFT) and the Convolution Theorem. The Convolution Theorem states

$$F(S \star \Psi) = F(S)F(\Psi)$$

so

$$S \star \Psi = F^{-1}(F(S)F(\Psi))$$

where $F(\bullet)$ denotes the Fourier transform and the multiplications taking place on the right hand side of the two equations above are *element-by-element.*

3.3.1 Parameters Comprising Θ space

Θ is the parameter space for the galaxy model we use, each member of this space is a 12-dimensional vector specifying a galaxy image. Here is a summary list of the 12 parameters, a brief description, and their units:

F	total integrated flux of galaxy ($erg \cdot cm^2/sec$)
B/T	bulge-to-total flux. Ratio of bulge integrated flux to F. i.e., $\frac{F_b}{F_b+F_d}$ ($unitless$).
μ_x	the x offset of the galactic center from the center of the image ($pixels$).
μ_y	the y offset of the galactic center from the center of the image ($pixels$).
sky	sky background offset ($flux/cm^2$). The DC component of the image.
s	Sersic index. A determinant of bulge shape that is fixed to the value $\frac{1}{4}$ for all experiments ($unitless$).
r_e	bulge scale length. Half-light radius of the bulge ($pixels$).
r_d	disk scale length. Scale radius of the disk ($pixels$).
ϵ_b	apparent bulge ellipticity ($unitless$).
γ_{inc}	disk inclination ($degrees$). Rotation toward the viewer.
γ_b	bulge angle of rotation ($degrees$). Clockwise rotation relative to the viewer.
γ_d	disk angle of rotation ($degrees$). Clockwise rotation relative to the viewer.

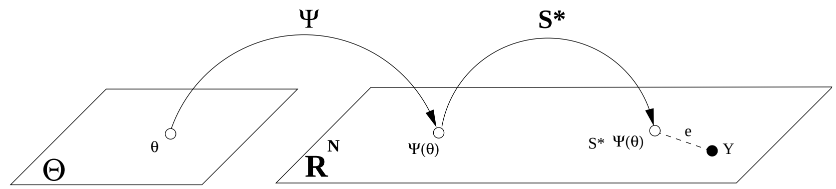


Figure 3.8: . Relationships within $\{\Theta, \Psi, S, Y\}$. Galaxies are generated by taking a point from parameter space Θ and mapping it via Ψ into image space, where it is then convolved with S, resulting in another point in image space. Input image Y is assumed to have been generated by this process and then had a Gaussian noise vector $\boldsymbol{e}$ added to it.

The space of Θ is constrained by certain simple linear inequalities, such as a requirement that the total flux must be nonnegative, the bulge fraction must be between 0 and 1, the ellipticity must be between 0 and 1, and that all radii must be greater than 0.

3.4 The Nonlinear Regression Problem

The inverse problem is the model $\mathcal{M}$, plus a target image Y, which is the tuple $\{\Theta, \Psi, S, Y\}$wherein

- Y or $\boldsymbol{y}$ is the image to be fitted, a square matrix with N total pixels. The image may have been subjected to noise added either before or after convolution with S.

We have no control over any of these elements. Figure 3.8 illustrates the relationships among the members of $\mathcal{M}$. The goal of the *inverse* problem is to take them and find a vector $\boldsymbol{\theta} \in \Theta$ that, when run through the forward model, produces an image close to Y. By 'close' we mean to minimize the sum squared error between the two images:

$$\chi^2(\boldsymbol{\theta}, Y) = \sum_i^n \sum_j^n (S \star \Psi(\boldsymbol{\theta}) - Y)_{ij}^2$$

Equivalently, we want to minimize the Euclidean distance between the vector $\boldsymbol{y}$ and $\overrightarrow{S \star \Psi(\boldsymbol{\theta})}$. This distance is also known as the magnitude of the error vector $\boldsymbol{e}$,

$$\sqrt{\chi^2(\boldsymbol{\theta}, Y)} = \|\boldsymbol{e}\| = \left\|\overrightarrow{S \star \Psi(\boldsymbol{\theta})} - \boldsymbol{y}\right\|$$

Another way to state the problem of minimizing this error function is that of maximizing the *variance explained.* Variance explained is

$$\mathsf{variance\,explained}(\boldsymbol{\theta}, Y) = 1 - \frac{\chi^2(\boldsymbol{\theta}, Y)}{\|\boldsymbol{y}\|^2} \tag{3.9}$$

In this paper, we will mainly refer to the error in terms of Euclidean distances when working with Principal Components Analysis, linear regression, and other geometrically-inspired discussions. However, variance explained and sum squared error will be useful in areas where signal-as-image interpretations are more natural, such as with 2-dimensional convolution.

Inverting the Model $\mathcal{M}$

We will denote as the manifold $\mathcal{G}$ the set of images that $\mathcal{M}$ can create in $\mathbb{R}^N$, $\mathcal{G} = \{S \star \Psi(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}$. Since Θ is a 12-dimensional space, $\mathcal{G}$ can at most occupy 12 dimensions of $\mathbb{R}^N$. To illustrate this geometrically: if the images are 32×32, then $\mathbb{R}^N$ can represent images of human faces, trees, handwriting, textures, abstract art, and an immensely larger number of meaningless 'noise' images. The images of galaxies from the manifold G will be a wisp in this hulking 1024-dimensional space of possibilities.

Inverting the forward model is a question of mapping any image in $\mathcal{G}$ back into the Θ space. If the input image is not exactly on the manifold, then the nearest point (in the Euclidean sense) on the manifold will be used.

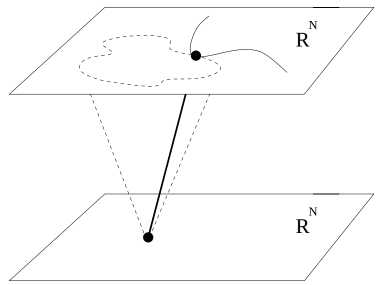


Figure 3.9: The many-to-one difficulty with deconvolution. Suppose the mapping is one of convolving images with a PSF. Many different images can produce an identical blurred image (bottom point). However, if we know that the original image came from a restricted subset of $\mathbb{R}^N$ (upper curved line segment), then we can perhaps reduce the possibilities to a narrow range.

Deconvolution

Deconvolution is basically ill-posed [55]. It does not fulfill the three Hadamard conditions of existence, uniqueness, and stability. Due to the loss of information involved in blurring an image, there is often no unique solution, so some priors have to be employed.

Because deconvolution does not fulfill the condition of stability, a small error in the data can cause a large error in the solution. Deconvolution in general is thus very intolerant to noise or error. Sources of such noise include data errors occurring after the optical image is formed, errors in the estimate of the PSF, variations in the real PSF across the image field, Poisson-distributed photon noise in imagery of faint sources, or image effects caused by signals (cosmic rays) impinging on the detector that are not subject to the blurring effects of the optical system.

So, how can we overcome this inverse problem? We have the benefit of knowing what we *expect* to see. Inverting a many-to-one function is not impossible if you can rule out most of the 'many' candidates as not being in the original domain. One way this can be done is to trace a large number of paths from Θ all the way through the convolution process. If an incoming image is close to the terminus of one of these paths, then we assign the corresponding $\boldsymbol{\theta}$ to the image. Figure 3.9 illustrates a realization of this.

3.5 Another Approach: GIM2D

One current technique, Galaxy IMage 2D (GIM2D) [52], uses the Metropolis algorithm [30] to fit surface brightness profiles to data. The method is a cousin of Simulated Annealing (SA) discussed in Section 2.3.2. The choice of the SA family of algorithms is meant to address the many local minima found in these types of problems. Quasi-Newton descent and other types of 'local' optimizers tend to quickly get stuck in a local minimum and progress no further. SA-type algorithms sacrifice speed for the more global nature of their search since they evaluate many more steps.

GIM2D first roughly bounds the search space based on the first and second moments obtained from the input image. This gives an initial starting point $\boldsymbol{\theta}$, and starting temperature vector T. GIM2D further refines the search's temperature T and parameter vector $\boldsymbol{\theta}_0$ by randomly sampling the current search space according to the following scheme:

$$\Delta\boldsymbol{\theta}_i = \boldsymbol{\theta}_{0,i} + (u - \frac{1}{2})T_i \tag{3.10}$$

where u is a uniform random deviate from $[0, 1]$ and T_i is the 'Metropolis temperature' of parameter i. This volume is sampled uniformly randomly n_{sample} times (on order of 400), and the best point found becomes the new center $\boldsymbol{\theta}'$. The search volume is then 'cooled' and shrunken by adjusting the T parameter to $T_i' = T_i/(n_{sample})^{1/n}$ where $n = 12$ is the number of parameters in the galaxy image model.

The third stage involves a detailed search of the current volume. At each step, the likelihood of the current $\boldsymbol{\theta}$ point is evaluated as

$$P(\boldsymbol{\theta}) = P(Y \mid \boldsymbol{\theta}, \Psi)$$

being probability of the data Y given the parameters $\boldsymbol{\theta}$ and the model Ψ. The current point is then perturbed by an amount determined by the temperature of the system. A perturbation step is

$$\Delta\boldsymbol{\theta}_i = (u - \frac{1}{2})T_i$$

where u is a uniform random number from $[0, 1]$. This trial point is evaluated to give its likelihood as well. If $P_{trial} > P_0$, then the trial step is immediately accepted. Otherwise, the step is accepted with probability P_{trial}/P_0. As the algorithm converges to a minimum, this algorithm will start rejecting a larger proportion of its steps, inefficiently exploring its space. This is avoided by adjusting the temperature T such that the number of trial steps accepted is about equal to the number rejected.

The trial steps are generated in such a way as to attempt to follow the local topology of the search space. Assuming a quadratic approximation to the local topography, $P(\boldsymbol{\theta}) = c + b^T\boldsymbol{\theta} + \boldsymbol{\theta}^T Q^T Q\boldsymbol{\theta}$, then the covariance matrix Q^TQ will sufficiently describe its shape. Given this, the steps are generated by $\Delta\boldsymbol{\theta} = Q\bar{u}$ where $\bar{u}$ consists of uniformly generated points from $[-1, 1]$. The matrix Q is updated every fifty iterations. This is similar to the technique often used in Simulated Annealing (see Section 2.3.2.)

The algorithm stops when the change in likelihood over 100 iterations is less than 3σ of the observed likelihood fluctuations.

3.6 Summary

In this chapter we have considered the surface brightness distribution of different types of galaxies. We have noted that the surface brightness distribution of almost all galaxies can be well described as being a combination of a spheroidal bulge and a flat disk. Early observations and fits indicate that the bulge follows the de Vaucouleurs $r^{1/4}$ law while the disk has an exponential distribution. Both these laws are empirical in nature. From the point of view of understanding galaxy morphology, it is very important to be able to fit these components; the causes of galaxy morphology are poorly understood at this time, and there is a huge reservoir of untapped image data. The next chapter develops an algorithm that can use this galaxy image model to fit real galaxy images.

Chapter 4

GMorph

Here we introduce an algorithm addressing the galaxy morphology task. The algorithm must be able to invert the galaxy image model and to deconvolve images that have been blurred by a PSF, and it must be able to do so quickly. After much trial and error, the most successful algorithm type was found to be Memory-Based Reasoning (MBR). In this case, memory-based means that an initial random population of synthetic images of galaxies are generated and then stored. When a query image comes in, it is compared to this population, and the nearest image's parameters are returned. This creates a general mapping from galaxy image space to parameter space by organizing and recalling from many specific precomputed mappings.

The above approach is a straightforward application of MBR. The difficulty, as usual, lies in the representation of the example space (here image space.) We have the problem of a large number of examples compounded by a very high-dimensional space. GMorph attacks both fronts using two techniques: decomposition of the image model and eigengalaxies.

In reducing the impact of the number of examples, GMorph takes advantage of the fact that the galaxy image model Ψ is composed of a linear combination of two models: a disk and a bulge. GMorph only stores examples of pure disks and bulges, recombining them later while searching the space. Reducing the

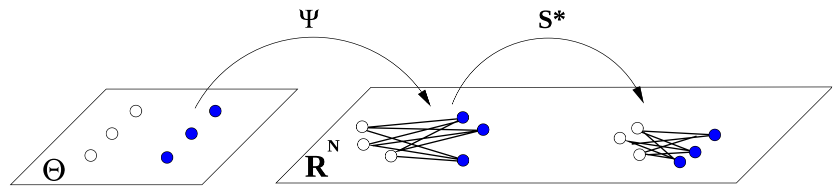


Figure 4.1: Factored sampling of the model space. The open circles are pure disks and the filled circles are pure bulges. All linear combinations of two endpoint images are depicted as a line segment. (In reality they are planes. See Figure 4.2.) From the six images sampled, an infinite number of sample images can be produced, which also cover a much larger area of the image space.

number of dimensions of the search space is achieved via eigengalaxies. The eigengalaxies represent a subspace of the image space $\mathbb{R}^N$ that is much more parsimonious in the number of dimensions needed to represent a galaxy image, and is thus faster to search.

Why go with MBR over, for instance, a numerical technique like Quasi-Newton descent? Our initial approach to the problem actually was Quasi-Newton descent with multiple starts. However, we increasingly found that the problem of local minima was severe. Many restarts were required in order to achieve a satisfactory level of certainty about the minimum, and local search was computationally expensive. Thus the location of the restart points was found to be critical to success.

In our algorithm, the initial sampling of the space determines the restart locations. This sampling a) has a large effect on the quality of the local minima, and b) shifts the computational burden off-line (since sample images can be precomputed.) Increasing emphasis was therefore placed on this valuable initial survey of the space, until eventually the best algorithm seemed to be to sample the space at a high resolution and return the best sample point, i.e., MBR. This is the form of the algorithm presented here.

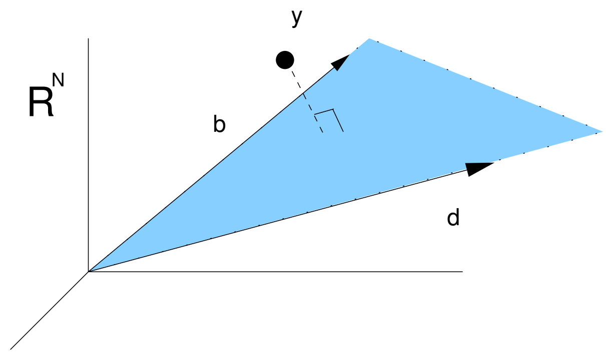


Figure 4.2: Bulge/disk decomposition. Vector $\boldsymbol{b}$ is the bulge image, $\boldsymbol{d}$ is the disk image. The shaded wedge-shaped plane that is bounded by the two vectors is the space of all images that this particular disk/bulge combination can represent. The input image is the point $\boldsymbol{y}$, and its best fit to $(\boldsymbol{d}, \boldsymbol{b})$ is its projection onto the shaded plane, obtained by Equation 4.2.

4.1 Model Decomposition

The algorithm takes advantage of the fact that the model can be decomposed into a linear combination of disk and a bulge. Thus only pure disk and pure bulge samples are taken from the manifold, on the premise that they can be recombined to create a much larger population of bulge/disk hybrid galaxies. A catalog of disk images and bulge images is created. This consists of sampling most of the parameter space Θ uniformly randomly[1] n_d times for pure disks (by fixing B/T to 0), and using those parameters as inputs to the model Ψ. The same is done for some number[2] n_b of pure bulges (obtained by fixing B/T to 1.) The results are organized into two sets of images B and D, and the parameters that produced them.

$$B = [\boldsymbol{b}_1,\ \boldsymbol{b}_2,\ \cdots\ \boldsymbol{b}_{n_b}]$$

$$D = [\boldsymbol{d}_1\ \boldsymbol{d}_2\ \cdots\ \boldsymbol{d}_{n_d}]$$

[1] Total flux F is fixed at 1, and μ_x and μ_y are fixed at 0.

[2] Typically 4000. We assume for simplicity here that the same number of disks as bulges are used.

All disks and bulges can be legitimately combined in pairs of a disk and a bulge, so these two sets can create $n_d \times n_b$ different combinations. Since a single combination can create any image $\alpha \boldsymbol{d}_i + \beta \boldsymbol{b}_j$, for any two positive scalars α and β, each combination can produce an infinite number of images by varying α and β. Figure 4.1 illustrates this. Geometrically speaking, $\boldsymbol{d}_i$ and $\boldsymbol{b}_j$ are two vectors in image space, defining the boundaries of a plane *within* which any point is a valid image[3].

In order to use these disk/bulge combinations in a memory-based application, we must define a distance metric between a disk/bulge pair and a data image. The shortest distance between the data image and the disk/bulge plane is a projection like the one in Figure 4.2. This distance is calculated via linear regression, which also determines the pair's best (w.r.t. χ^2) total flux parameter, F, and bulge-to-total ratio, B/T.

$$Z_{ij} = [\boldsymbol{d}_i \, \boldsymbol{b}_j] \tag{4.1}$$

is a matrix whose two columns hold the i^{th} disk image and the j^{th} bulge image. They are the bases for the space that $\boldsymbol{y}$ is projected onto in the following step.

$$\boldsymbol{y}^*_{ij} = Z_{ij}(Z^T_{ij} Z_{ij})^{-1} Z^T_{ij} \boldsymbol{y} \tag{4.2}$$

$$\chi^2_{ij} = \left\| \boldsymbol{y} - \boldsymbol{y}^*_{ij} \right\| \tag{4.3}$$

Since negative flux violates a constraint in the model, $\boldsymbol{y}^*_{ij}$ must be composed of some *nonnegative* amount of each component. In the case that either component has a negative contribution, then the fit is performed again with the offending component removed from Z_{ij}, thus forcing $\boldsymbol{y}$ to be projected onto the nearest point on the ray defined by the other component. This will be the least-squares projection onto the space defined by positive combinations of $\boldsymbol{d}_i$ and $\boldsymbol{b}_j$.

[3] Not a full plane, of course, but a half-plane, quarter-plane, or some other fraction of a plane. See Figure 4.2.

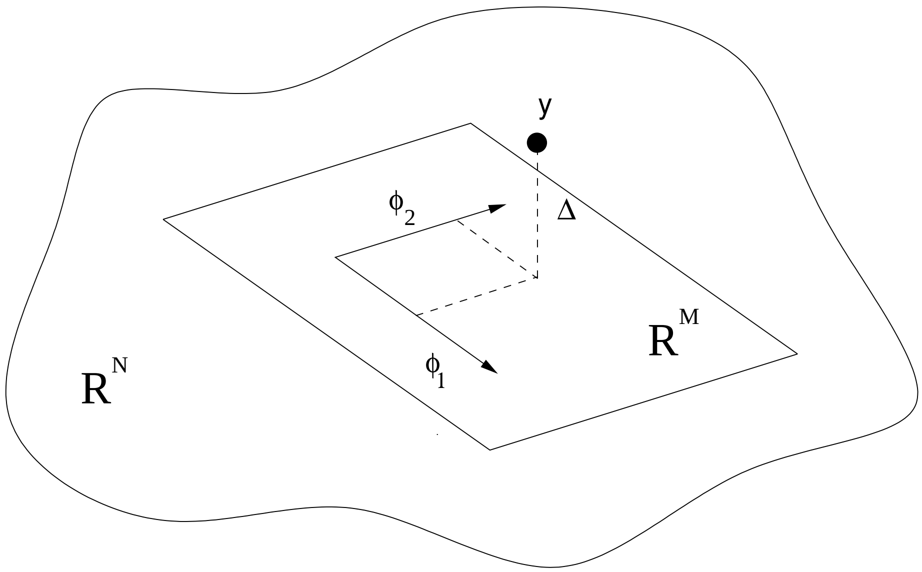


Figure 4.3: Eigenspace projection is a linear subspace of $\mathbb{R}^N$. The bases ϕ_i are perpendicular with a length of 1. In this case, the first and second eigencoordinates would be $\phi_1^T \boldsymbol{y}$ and $\phi_2^T \boldsymbol{y}$, where the dashed lines intersect the coordinate system. The vector $\boldsymbol{\Delta}$ is the reconstruction error.

At this point, we theoretically have only to calculate χ^2_{ij} for all combinations of disks and bulges, and select the combination with the lowest error (i', j'). The image $\boldsymbol{y}^*_{i'j'}$ will be our best fit. Two major obstacles have yet to be overcome, however: computational speed and the PSF.

4.2 Eigengalaxies

One difficulty with working in an image space $\mathbb{R}^N$ is the large computational load in terms of storing and manipulating points. Since the model $\mathcal{M}$ can at most create images that occupy 12 dimensions of $\mathbb{R}^N$, there is a great potential gain in efficiency by reducing the dimensionality. Principal Components Analysis (PCA) is a method that can be used to determine the linear subspace in which most of the data variance resides. It is also known as the Karhunen-Loeve Transform [27] with only the first K basis functions used. The first K princi-

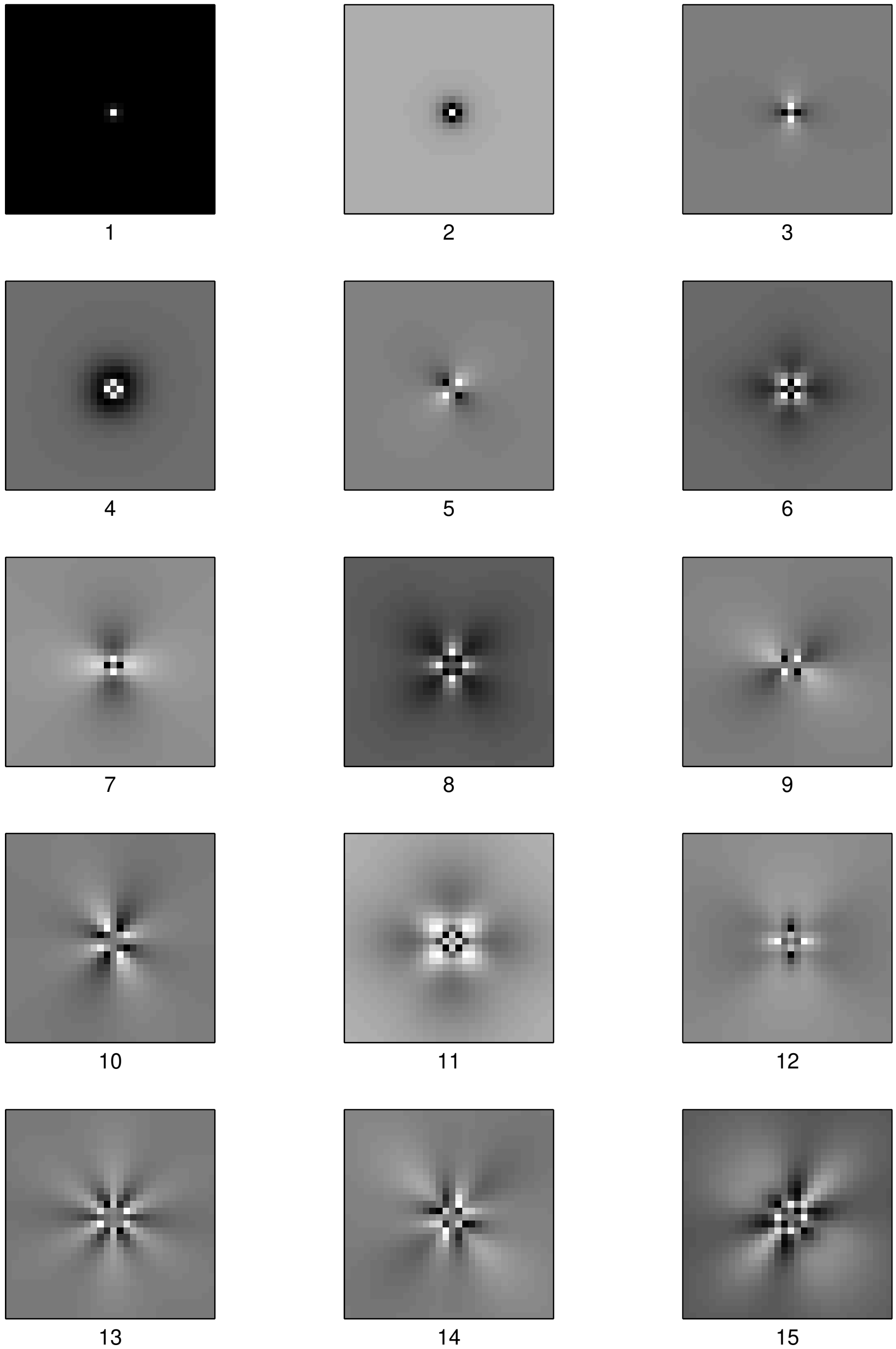


Figure 4.4: The first 15 eigengalaxies obtained by the procedure in section 4.2.

pal components of a dataset are defined to be the eigenvectors of the dataset's covariance matrix with the K highest eigenvalues. The images first have their mean subtracted[4] and are then placed column-wise in matrix U. Since the columns represent images and the rows represent pixels, UU^T will be the covariance matrix among the pixels. Finding all solutions to the equation

$$UU^T\boldsymbol{\nu} = \lambda\boldsymbol{\nu}$$

and taking only the vectors $\boldsymbol{\nu}$ with the K highest eigenvalues λ will produce the first K principal components. The span of these principal components will maximize the amount of variance containable in K basis vectors.

PCA has been very successful in image processing [32,38,46]. Some relevant advantages of an eigenspace are that it

- Compactly represents the data. It explains the greatest amount of variance for a given number of bases.
- It is orthogonal, so Euclidean distances are meaningful.
- Orthogonality also means that projection of points into the space is cheap. Projection otherwise entails inverting a $K \times K$ matrix.

In order to determine the eigenspace of the model$\{\Theta, \Psi, S\}$, we use a large number[5], n_b, of bulge images and n_d disk images which were generated in Section 4.1. All the images are used to determine a covariance matrix for the pixels by stacking the mean-normalized images as column vectors in a single $N \times (n_d + n_b)$ matrix

$$U = [D\ B] = [\boldsymbol{d}_1,\ \boldsymbol{d}_2,\ \cdots\ \boldsymbol{d}_{n_d},\ \boldsymbol{b}_1,\ \boldsymbol{b}_2,\ \cdots\ \boldsymbol{b}_{n_b}]$$

The individual elements of the covariance matrix $(UU^T)_{ij}$ represent the covariance between pixel i and pixel j in all the images. The first K eigenvectors

[4]Although not necessary for PCA, we also ensure that each image has unit variance (hence equal weight.)

[5]Typically 4000

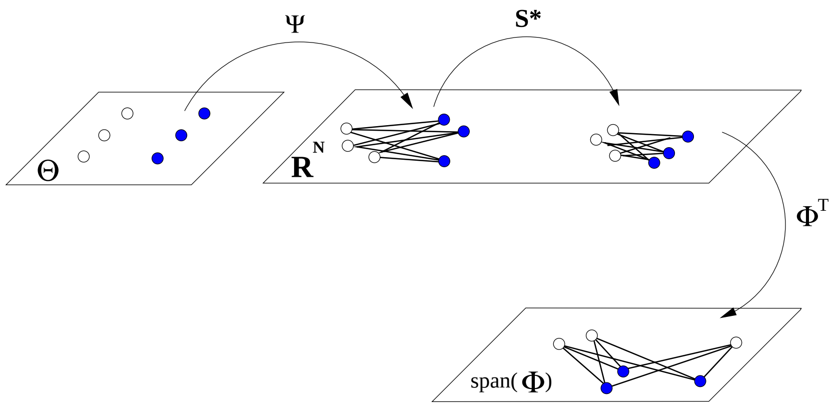


Figure 4.5: Adding an eigenspace. The new space $\mathsf{span}(\Phi)$ is a $K-$dimensional subspace of $\mathbb{R}^n$. The same linear relationships apply in this eigenspace as did in the image space.

of the covariance matrix UU^T are arranged as the columns of the 'eigenspace matrix' Φ. Hence Φ is an $N \times K$ orthogonal matrix such that

$$\Phi^T : R^N \longrightarrow R^K$$

and the span of the columns of Φ will be referred to as an eigenspace. Note that Φ^T is a projection such that

$$\tilde{\boldsymbol{y}} = \Phi^T \boldsymbol{y}$$

$$\left[\widetilde{d}_1, \widetilde{d}_2, ..., \widetilde{d}_{n_d}\right] = \Phi^T \left[\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_{n_d}\right]$$

where eigencoordinates are denoted by $\widetilde{\bullet}$. As is shown, Φ^T maps images in $\mathbb{R}^N$ to points in eigenspace. Since Φ^T is orthogonal, the transpose Φ maps points in eigenspace back to points in $\mathbb{R}^N$. Unless the point originally came from the eigenspace, though, there will be some *reconstruction error* Δ due to the fact that only K dimensions were retained during the Φ^T projection.

$$\Delta = \left\|\boldsymbol{y} - \Phi\Phi^T \boldsymbol{y}\right\|$$

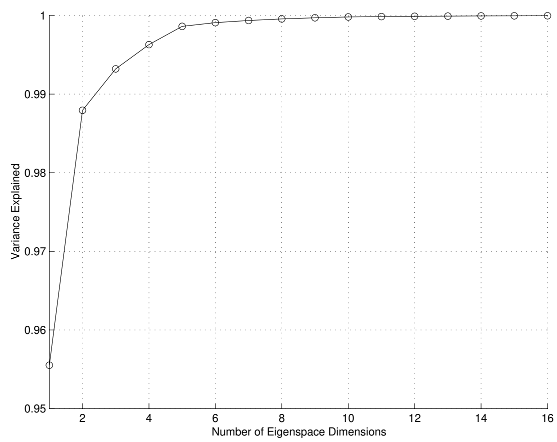


Figure 4.6: Percent of synthetic galaxies' variance that is explained by increasing the number of eigenvectors in Φ. Percent variance explained by the i-th eigenvector in Φ is obtained by $\frac{\lambda_i}{\sum_j^N \lambda_j}$.

The distance between the input image $\boldsymbol{y}$ and some image in the catalog $\alpha\boldsymbol{d} + \beta\boldsymbol{b}$ can be computed as

$$\chi^2(\boldsymbol{y}, \alpha\boldsymbol{d} + \beta\boldsymbol{b}) = \left\| \Phi^T\boldsymbol{y} - \left(\alpha\Phi^T\boldsymbol{d} + \beta\Phi^T\boldsymbol{b}\right) \right\|$$

A possible refinement would be to create the eigenspaces using actual galaxy images to see if this produces better results. This was not pursued here for several reasons, primarily because of the difficulty in aligning the centers of galaxies and difficulty in selecting representative galaxies. It is also unclear whether this would be a superior approach, since the reduction in reconstruction error for $\boldsymbol{y}$ would be offset to some extent by an increase in the reconstruction error for B and D. Finally, using synthetic galaxies it is easy to control the properties of a catalog, e.g., the distribution of parameters.

4.3 Local Eigenspaces

4.3.1 PSF-Localized Eigengalaxies

We have assumed up to this point that the PSF is the same for all problems, allowing us to build a catalog tailored to that exact PSF. We now relax that assumption, so the PSF is different for every galaxy. One direct solution to this problem is to deconvolve Y using the given S. However, deconvolution suffers from instability in the presence of noise. Another approach would be to incorporate PSFs into a global eigenspace. This global PCA approach would be to sample the manifold$\{s \star \Psi(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta \wedge s \in \mathcal{S}\}$, where $\mathcal{S}$ is the set of all possible PSFs that one is likely to see. This global approach forces the eigenspace to be unnecessarily large.

Our approach is a slight modification called PSF-localized PCA. To deal with the different PSFs, we keep a small number of PSF-specific eigenspaces and of eigencoordinates $\left\{(S, \Phi, \tilde{B}, \tilde{D})_i\right\}_{i=1}^{n_s}$, where n_s is the number of PSFs that have been stored. The initial PSF population consists of a few Gaussians of varying standard deviation, to which PSFs are added during on-line operation. This approach of Local PCA is common in image processing [41].

These new catalogs are generated by taking the unblurred images in B and D and convolving each with the same PSF, $S[i]$, the results being the new matrices $B[i]$ and $D[i]$. These images are then used to create a new eigenspace $\Phi[i]$, which $B[i]$ and $D[i]$ are projected into to produce $\tilde{B}[i]$ and $\tilde{D}[i]$. This is repeated n_s times.

Now if we are given an S that matches $S[i]$, we can use $\Phi[i]$, $\tilde{B}[i]$, and $\tilde{D}[i]$ to make the fit to $\boldsymbol{y}$. In general, we do not expect a perfect match, however. The effect of PSF mismatch is generally slight for small variations (see Figure 4.8), but the effect can be minimized further with the following procedure. Figure 4.9 describes the general approach.

1. Find the closest (see below) PSF in memory, $S[i^*]$, to S.

2. Deconvolve S with it, i.e., solve for ΔS in the relation $S = S[i^*] \star \Delta S$.

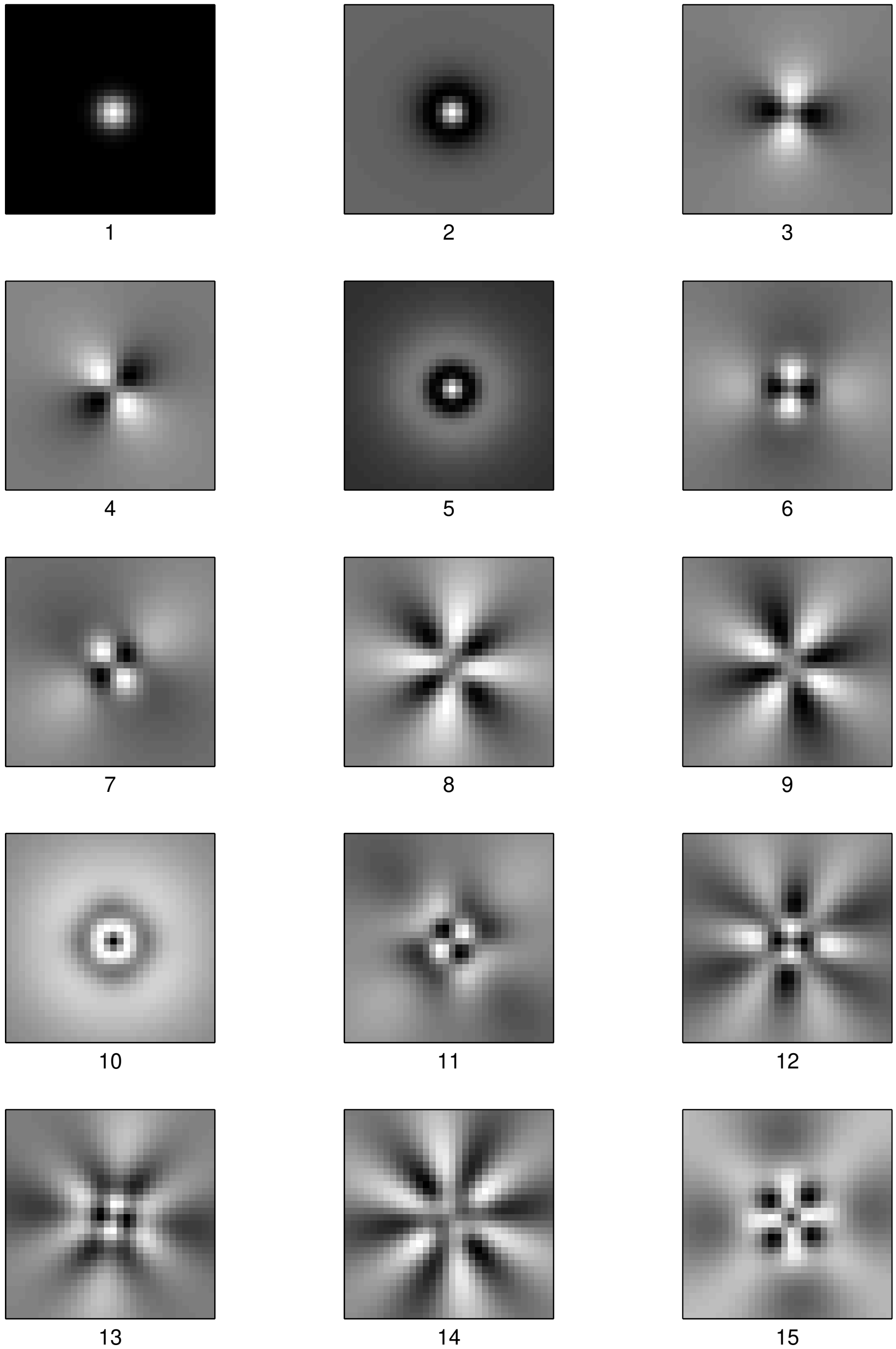


Figure 4.7: The first 15 eigengalaxies obtained from galaxies convolved with a Gaussian PSF with 2-pixel standard deviation.

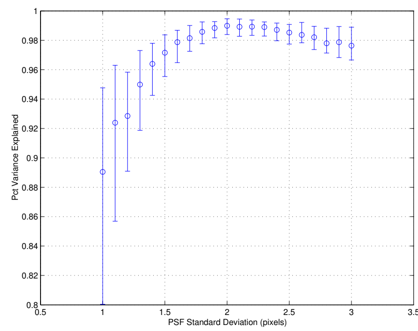


Figure 4.8: Effect of PSF mismatch. The catalog's one PSF is a Gaussian with 2-pixel standard deviation. For each point on the plot, 100 simulated galaxy images were generated by the process described in Section 4.6.2 using a PSF which is a Gaussian of varying standard deviation (x axis). The variance explained achieved with the catalog is plotted on the y axis. Circles are the median values, error bars are the 25^{th} and 75^{th} percentiles.

3. Convert each column of $\Phi[i^*]$ into an image, and convolve it with ΔS. Reassemble the results as the columns of a new matrix $\Phi_{\Delta S}$. (The eigengalaxy images $\phi_\bullet$ below are from the columns of $\Phi[i^*]$.)

$$\Phi_{\Delta S} = \left[\overrightarrow{\Delta S \star \phi_1}, \ \overrightarrow{\Delta S \star \phi_2}, \ \cdots \ \overrightarrow{\Delta S \star \phi_K} \right] \tag{4.4}$$

   This is a different method for creating a blurred 'eigenspace' compared to those used to create $\Phi[i^*]$ in the first place. There, each image in B and D was convolved with a PSF, then the principal components of those images were extracted. Here, we take an already existing eigenspace and just convolve the PSF with the K eigengalaxies in it. This is a quick way to create a new 'eigenspace', but it warps the basis vectors such that they are no longer orthogonal. Hence the final step:

4. To find $\boldsymbol{y}$'s eigencoordinates in $\Phi[i^*]$, we use $(\Phi_{\Delta S}^T \Phi_{\Delta S})^{-1}$ to undo the warping caused in Equation 4.4.

$$\tilde{\boldsymbol{y}}[i^*] = (\Phi_{\Delta S}^T \Phi_{\Delta S})^{-1} \Phi_{\Delta S}^T \boldsymbol{y} \tag{4.5}$$

we can see that the assumption is that $\boldsymbol{y}$ can be adequately described by $\Phi_{\Delta S}$.

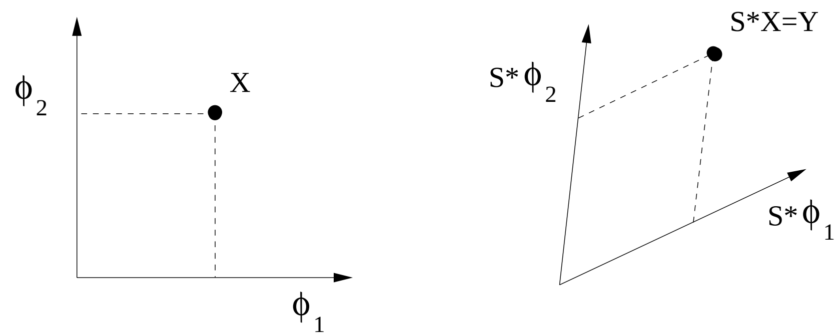


Figure 4.9: The final PSF correction. Assuming that the undistorted image $\boldsymbol{x}$ lies in eigenspace Φ (on left), then $\boldsymbol{y}$ in space Φ_S will have the same coordinates.

If indeed $\boldsymbol{y} = \Phi_{\Delta S}\tilde{\boldsymbol{y}}$, then $(\Phi_{\Delta S}^T \Phi_{\Delta S})^{-1}(\Phi_{\Delta S}^T \Phi_{\Delta S})\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{y}}$ implies Equation 4.5.

Closeness in this context means that $S[i^*]$ is the PSF that minimizes $\chi^2(S, S[i^*] \star \Delta S) + \chi^2(S[i^*], S[i^*] \star \Delta S)$. We want a ΔS that can be convolved with an existing PSF in such a way that the distance to the data PSF is minimized *and* the distance from the existing PSF is minimized.

4.3.2 Offset-Localized Eigengalaxies

Another assumption so far has been that all galaxies have no sub-pixel offset, i.e., that μ_x and μ_y are integers. We now relax that assumption. Again, the global PCA approach would dictate creating a single eigenspace from the manifold $\{S \star \Psi(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta \wedge -1 \leq \mu_x \leq 1 \wedge -1 \leq \mu_y \leq 1\}$. Most of the variance in this larger manifold is attributable to the subpixel offset and not to the morphology of the galaxy, so it is less useful than local PCA. For offset-localized PCA, we select a number of x- and $y-$offsets, n_x and n_y, to use to create a set of subpixel offsets, e.g., $\{(0,0), (\frac{1}{n_x},0), (0,\frac{1}{n_y}), (\frac{1}{n_x},\frac{1}{n_y}),\ldots (\frac{n_x-1}{n_x},\frac{n_y-1}{n_y})\}$. For each separate offset j, we generate a new Φ_j, $\tilde{B}_j$, and $\tilde{D}_j$ in the manner used in Section4.1. Thus, we keep a set of eigenspaces and eigencoordinates $\left\{(\Phi, \widetilde{B}, \widetilde{D}, S)_{ij}\right\}_{i=1,j=1}^{n_s, n_x \times n_y}$, where n_s is the number of different PSFs that have been stored, and $n_x n_y$ is the number of subpixel offsets that have been stored. This results in a total of $n_s n_x n_y$ different catalogs, each requiring approximately

$\mathcal{O}(K(n_d + n_b) + K^2)$ in storage.

4.3.3 Picking a Local Eigenspace

In order to select *which* eigenspace to work in, GMorph finds which of these eigenspaces is closest to the input image. The algorithm thus finds the distance from $\boldsymbol{y}$ to $span(\Phi)$ for all possible translations of $\boldsymbol{y}$. These distances,

$$dist_{ij}(\boldsymbol{y}, \Phi) = \left\| \boldsymbol{y} - \Phi\Phi^T \boldsymbol{y} \right\| \tag{4.6}$$

are computed via the Fast Fourier Transform (FFT). Since convolution by definition (Equation 3.8) computes the dot product between two signals at every offset, we use FFT to convolve $\boldsymbol{y}$ with every image in Φ, yielding all the possible projections of $\boldsymbol{y}$ onto Φ. After $n_x n_y K$ different convolutions, it is straightforward to just select the translation and the eigenspace with the smallest distance. This selects μ_x and μ_y. Assuming that the image $\boldsymbol{y}$ was originally in $span(\Phi)$ and then corrupted by Gaussian error, this will be the Maximum Likelihood (ML) choice.

4.3.4 Masking Eigengalaxies

Often the data image contains sources of illumination that are not relevant to the galaxy of interest. Nearby galaxies and cosmic rays will interfere with the fit if the model has to try to explain them. A mask image is often included to deal with this. The mask is an image that flags which pixels should be ignored. During the development of GMorph, several approaches to this problem have been explored. Initially, setting the bad pixels to the value of the sky was a workaround, but it was a source of bias because parts of the galaxy of interest were often masked. The next technique was to replace the masked pixels with their most likely value at the time, and to iterate between fitting this new image and replacing the masked pixels afresh. This method suffered from local minima and added about 20% to the running time per galaxy. The final and current

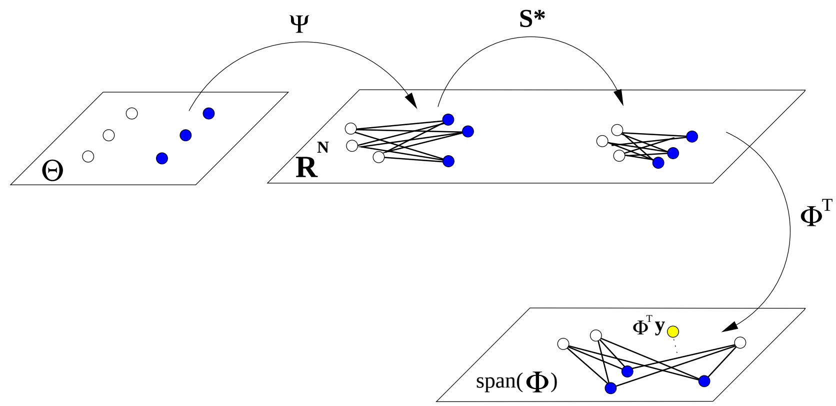


Figure 4.10: Finding the nearest neighbor. In the eigenspace $\mathsf{span}(\Phi)$, from the transformed input image $\Phi^T \boldsymbol{y}$, find the closest linear combination of a known disk and known bulge.

method has been the on-line creation of temporary *masked eigengalaxies.* Once created, the algorithm treats them as if they were normal eigengalaxies and finds the best fit.

$$\Phi_{\text{mask}} = [\mathsf{mask}(\phi_1),\ \mathsf{mask}(\phi_2),\ \cdots\ \mathsf{mask}(\phi_k)]$$

$$\tilde{\boldsymbol{y}}' = (\Phi_{\text{mask}}^T \Phi_{\text{mask}})^{-1} \Phi_{\text{mask}}^T \boldsymbol{y}$$

Φ_{mask} is created by taking each of the eigengalaxies in Φ and setting the masked pixels to 0. The eigencoordinates $\tilde{\boldsymbol{y}}'$ can now be used as normal coordinates in the space Φ (not Φ_{mask}). At no point in this process is it necessary to recalculate an eigenspace.

4.4 Finding the Nearest Neighbor

After the eigenspace has been selected and the input image has been projected into the eigenspace, then the search for a nearest neighbor can begin. The setting for this search is illustrated in Figure 4.10The search could be accomplished

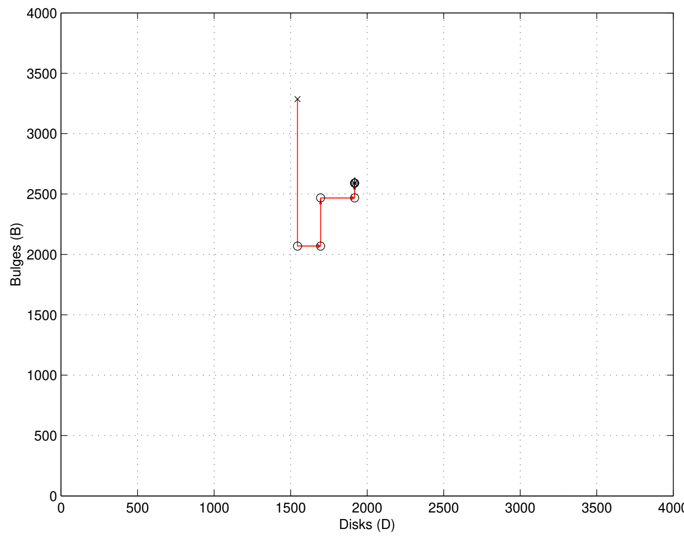


Figure 4.11: Coordinate search example. The bulges and disks are roughly ordered by radius size, but otherwise have no intrinsic ordering. The search begins at the $\times$, with disk #1550 and bulge #3300. The search proceeds in the bulge direction and finds a minimum with bulge #2050. The best disk is then found to be #1700, and so on. The following is the full search trajectory: (1550,3300),(1550,2050),(1700,2050), (1700,2450) ,(1900,2450),(1900,2600). The final point is a local minimum for this search.

by an exhaustive search of all bulge-disk combinations[6]. However, we can avoid the exhaustive search and save considerable time by random sampling and coordinate search, as discussed in Section 2.2.1. This search starts by selecting a random bulge from $\tilde{B}$. Then this bulge is paired with all disks from $\tilde{D}$ and a χ^2 is calculated for each combination when optimized. The best combination becomes the new 'start point' for another 'step'. Now the disk is fixed while $\tilde{B}$ is searched for a better bulge. The process continues in this manner until no further improvement can be had. The search can be described by the following, where χ^2 is obtained by linear regression as shown in Section 4.1. At point i in the search, the following describes the trajectory:

[6]This exhaustive search has been implemented and found too expensive in terms of memory and time. Significant speedups were possible by means of caching of intermediate values such as $D^T B$, $D^T \boldsymbol{y}$, and $B^T \boldsymbol{y}$. The final time complexity was about an order of magnitude larger than the current technique.

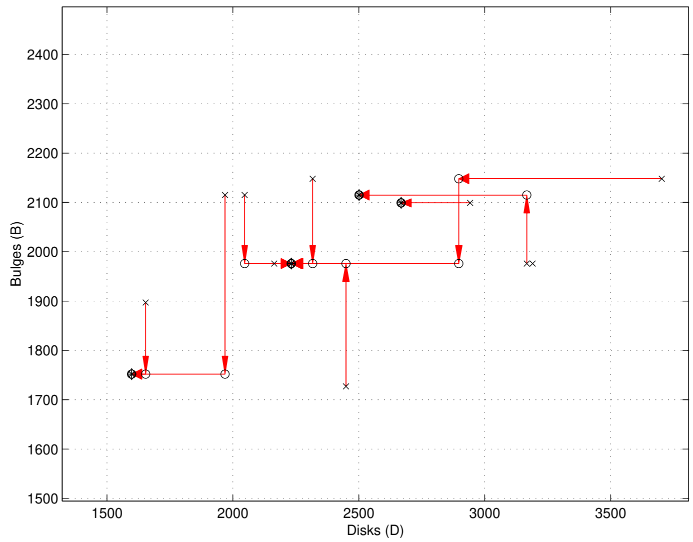


Figure 4.12: Coordinate search example. Multiple start points often enter the same trajectory. Multiple local minima (filled circles) also are frequent. The scale of the plot is twice that of Figure 4.11.

$$\tilde{d}_i = argmin_{\tilde{d} \in \tilde{D}} \, \chi^2(\tilde{d}, \tilde{b}_i, \tilde{\boldsymbol{y}})$$

$$\tilde{b}_{i+1} = argmin_{\tilde{b} \in \tilde{B}} \, \chi^2(\tilde{d}_i, \tilde{b}, \tilde{\boldsymbol{y}})$$

Examples of these trajectories are in Figures 4.11 and Figure 4.12. The process is guaranteed to converge because the search space is finite, and the sequence of pairs must always have a decreasing χ^2. So a local minimum will always be found. Usually this converges to a better minimum than a local search, such as quasi-Newton descent, since exploration always touches upon every individual bulge and disk at every other step. It is inherently non-local in this respect.

4.5 Algorithm Summary

Here we review in more detail the previous sections. There are two main parts: the off-line computation of the catalogs, and the on-line execution of the algorithm. The bulk of the code is written in Matlab, though the image-generating functions are in C.

I. Create Catalogs

The goal here is to create a set of 'catalogs' that are specific to different PSFs and subpixel offsets. Each catalog is a set of images, eigenspaces and the eigencoordinates of the images to use during the fitting process. The eigenspaces are $\Phi[i,j,k]$ and the eigencoordinates are $\tilde{B}[i,j,k]$ and $\tilde{D}[i,j,k]$ for $i = 1, 2, \ldots n_s$, $j = 1, 2, \ldots n_x$, and $k = 1, 2, \ldots n_y$.

1. **Create unblurred catalog**. The creation of $\Phi[0,0,0]$ require the following:

   (a) A catalog of disk images and bulge images is created which consists of sampling the parameter space Θ uniformly randomly inside the interval $F = 1$, $\mu_x = 0$, $\mu_y = 0$, $0 \le r_d \le 16$, $0° \le \gamma_{inc} \le 85°$, $0 \le \gamma_d \le 180°$, $0 \le r_e \le 16$,$0 \le \epsilon \le 0.7$, and $0° \le \gamma_b \le 180°$. Image size of 64×64 was used during the experiments. Images of pure disks were created by fixing B/T to 0 and the number of pure-disk images sampled was n_d. We do the same for n_b pure bulges (obtained by fixing B/T to 1.) The results are organized into two matrices B and D:

$$B = [\boldsymbol{b}_1,\ \boldsymbol{b}_2,\ \cdots\ \boldsymbol{b}_{n_b}]$$

$$D = [\boldsymbol{d}_1\ \boldsymbol{d}_2\ \cdots\ \boldsymbol{d}_{n_d}]$$

   (b) The eigenspace of K dimensions is calculated by putting all the images in a single matrix U

$$U = [B\ D]$$

Calculate the column-wise mean of U and subtract it from each column of U. Next we compute the eigenvectors of the covariance matrix UU^T.

$$UU^T\boldsymbol{\nu} = \lambda\boldsymbol{\nu}$$

Of the N eigenvectors that satisfy this equation, the K with the highest-valued eigenvalues are selected to be the columns of Φ. The products $\Phi^T B$ and $\Phi^T D$ give $\tilde{B}$ and $\tilde{D}$ respectively.

2. **Create offset-localized catalogs.** We also create $n_x \times n_y$ other 'offset' catalogs. These are created in the same way as the zero-offset catalogs in step (1), except now $(\mu_x, \mu_y) = (\frac{j-1}{n_x}, \frac{k-1}{n_y})$ for $j = 1, 2, \ldots n_x$ and $k = 1, 2, \ldots n_y$. These correspond to $\Phi[0, j, k]$.

3. **Create PSF-localized catalogs**. We also create n_s 'blurred' catalogs for each offset. The set of PSFs used have typically been Gaussian images of standard deviations of 0.5, 1, 1.5, 2, and 3 pixels[7]. The i-th PSF is referred to as $S[i]$. The blurred catalogs are created using the existing catalogs and convolving each of the images within them with a PSF. The catalogs are indexed by S and the subpixel offsets. So $\boldsymbol{b_t}[i, j, k] = \overrightarrow{S[i] \star \Psi(\boldsymbol{\theta_t})}$ where the μ_x and μ_y values of $\boldsymbol{\theta_t}$ are set to $\frac{j-1}{n_x}$ and $\frac{k-1}{n_y}$. After this we can throw away all the B, D, and U matrices except for those corresponding to the unblurred catalogs.

II. Compute Fit

1. **Get input.** Get the image Y, the PSF image S, and possibly mask image from user.

[7]More PSFs are added during run time, using the same procedure described here.

2. **Check input.** Make sure Y, S, and mask are well-formed.

   (a) Ensure that Y, S, and the mask (if present) have equal dimensions. If they do not, then either pad them with zeros or crop to match.

   (b) Ensure that S satisfies Equations 3.6 and 3.7, i.e., S has no negative values and sums to one. If it does not, then report error.

   (c) Ensure that the mask only has values of 0 and/or 1

   (d) Translate S so that its center pixel is in the upper left corner. This is a convenience; if the PSF is in the center, convolving another image with S will result in blurring *and* shifting the image down and to the right. The center pixel of S is assumed to be the modal pixel, i.e., the pixel with the maximum value. The centroid could also be used here; the main requirement is that similar PSFs be treated in a similar manner.

3. **Find catalog with the best matching S.** The best match here is determined to be the $S[i]$ that minimizes the sum squared error. The PSF index of the catalog is determined by

$$i^* = argmin_{i=1,2,\ldots n_s} \sum_{j=1}^{n} \sum_{k=1}^{n} (S - S[i])^2_{jk}$$

4. **Determine if $\Phi[i^*]$ needs to be adjusted for PSF mismatch**. If the percent variance explained (Equation 3.9) of S by $S[i^*]$ is more than a predetermined threshold[8], do not compute a ΔS, so continue to the next step. Otherwise:

   (a) If the variance explained is less than another threshold[9], then create a new catalog for this S. See step I.3.

[8] We used 98% in our experiments.
[9] We used 90% here.

(b) Deconvolve S with $S[i^*]$, i.e., solve for ΔS in the relation $S = S[i^*] \star \Delta S$. Currently this is achieved via the Richardson-Lucy deconvolution algorithm [28] as implemented in Matlab 6.1.

(c) Convert the columns of $\Phi[i^*]$ into images, and convolve them with ΔS. Reassemble the results as the columns of a temporary space $\Phi_{\Delta S}$ as depicted in the formula below. (The images $\phi_\bullet$ are from the columns of $\Phi[i^*]$.)

$$\Phi_{\Delta S} = \left[\overrightarrow{\Delta S \star \phi_1},\ \overrightarrow{\Delta S \star \phi_2},\ \cdots\ \overrightarrow{\Delta S \star \phi_K}\right] \tag{4.7}$$

$$\Phi' = (\Phi_{\Delta S}^T \Phi_{\Delta S})^{-1} \Phi_{\Delta S} \tag{4.8}$$

this matrix will be used later to compute $\tilde{\boldsymbol{y}}$. During the rest of this fit, whenever a Φ is going to be used to transform $\boldsymbol{y}$, it must be used as Φ' instead, via Equations 4.7 and 4.8.

5. **Adjust for mask, if present.** In the same manner as the previous step, convert the columns of $\Phi[i^*]$ into images, then multiply each eigenimage element-by-element with the mask image. Reassemble the results as the columns of a temporary space Φ_{mask}. The mask multiplication process is denoted by the $\mathsf{mask}(\cdot)$ operation.

$$\Phi_{\text{mask}} = [\mathsf{mask}(\phi_1^*),\ \mathsf{mask}(\phi_2^*),\ \cdots\ \mathsf{mask}(\phi_k^*)]$$

$$\Phi'' = (\Phi_{\text{mask}}^T \Phi_{\mathsf{mask}})^{-1} \Phi_{\text{mask}}^T$$

as with the previous step, whenever Φ is used in the future, substitute Φ''.

6. **Find the integer offsets** $(\Delta x, \Delta y)$. First assume that the subpixel offsets (j, k) are zero. The task is to find the integer offset of $\boldsymbol{y}$ from $\Phi[i^*, 0, 0]$. This is done by finding the χ^2 for all $n \times n$ integer translations:

(a) Expand each of the K eigengalaxies in $\Phi[i^*, 0, 0]$ to match the size of Y by padding each eigengalaxy with zeros.

(b) Convolve each eigengalaxy ϕ in $\Phi[i^*_s, 0, 0]$ with Y, resulting in K $n \times n$ matrices which we call C_i where $i = 1, 2, \ldots K$. The value of one element in one of these matrices is equal to

$$c_{ijk} = (\phi_i \star Y)_{jk} = \phi_i^T \mathsf{shift}(\boldsymbol{y}, j, k) \tag{4.9}$$

this gives us the eigencoordinates $\tilde{\boldsymbol{y}}$ for every translation of Y. Specifically, the j, k-th element of matrix C_i gives the i-th eigencoordinate of $\tilde{\boldsymbol{y}}$ if Y is translated j, k pixels.

(c) In order to find the smallest reconstruction error for each of the eigencoordinates, we find the offset $(\Delta x, \Delta y)$ that minimizes

$$-\mathsf{shift}(\boldsymbol{y}, \Delta x, \Delta y)^T \Phi \Phi^T \mathsf{shift}(\boldsymbol{y}, \Delta x, \Delta y) \tag{4.10}$$

This can be calculated from the C matrices from (4.9) by simply squaring every element and adding all the matrices together. This will result in a single $n \times n$ matrix, of which element (i, j) is the value of (4.10) for offset (i, j).

7. **Hillclimb to find subpixel offsets** (j^*, k^*). We already have a good idea where the best overall offset is from the previous step; it is somewhere very close to $(\Delta x^* + 0, \Delta y^* + 0)$. Next find the χ^2 for the surrounding offsets $(\Delta x \pm \frac{1}{n_x}, \Delta y \pm \frac{1}{n_y})$. We start by setting (j, k) to $(0, 0)$. We then calculate

$$\tilde{\boldsymbol{y}}_{i^* j' k'} = \Phi[i^*, j', k']^T \mathsf{shift}(\boldsymbol{y}, \Delta x, \Delta y)$$

for all $j' \in \{\mathsf{mod}(j-1, n_x), \mathsf{mod}(j+1, n_x)\}$ and $k' \in \{\mathsf{mod}(k+1, n_y), \mathsf{mod}(k-1, n_y)\}$. Each χ^2 value is calculated via Equation 4.10. If a better value is found, update (j, k) accordingly and move to that position. Repeat until improvement stops. The final (j, k) values are (j^*, k^*). We now have determined which eigenspace to use:

$$\Phi^* = \Phi[i^*, j^*, k^*]$$

$$\tilde{B}^* = \tilde{B}[i^*, j^*, k^*]$$

$$\tilde{D}^* = \tilde{D}[i^*, j^*, k^*]$$

8. **Search Eigenspace for a match**

   (a) Randomly sample some number[10] of pairs of bulge and disk images from $\tilde{B}^*$ and $\tilde{D}^*$. Find the best fit of each pair to $\tilde{\boldsymbol{y}}$ via linear regression in the relation

   $$\tilde{\boldsymbol{y}} = \alpha \tilde{\boldsymbol{b}} + \beta \tilde{\boldsymbol{d}} + \chi^2$$

   using Equations 4.1 through 4.3. Sort the pairs by their χ^2 values.

   (b) Coordinate search the top 10 pairs from the previous step. A pair $(\tilde{\boldsymbol{d}}, \tilde{\boldsymbol{b}})$ will be a point in search space. A point's evaluation is the χ^2 value for the pair as determined via linear regression by the Equations 4.1 through 4.3. Starting from a point in the search space, search starts by randomly deciding to search in the 'disk direction' or the 'bulge direction'. Supposing that the bulge direction is chosen, this would entail trying out every bulge $\tilde{b}$ in $\tilde{B}$ with the current disk. The best resulting pair is kept as the new point. Subsequent searches alternate directions. When a search results in no better point or a point that has already been reached in a previous search, then the search terminates.

   (c) Return best pair found in all the searches as determined by the pair's χ^2 value.

[10] We used 40,000 in our experiments

4.6 Experiments

4.6.1 Data Fits

The data images are all taken from SDSS archives. The PSFs of the data images are not shown, but they are all approximately Gaussian with a standard deviation of about 2.2 pixels. Sample fits were made with a precomputed catalog containing 500 bulge and 500 disk images, with subpixel offsets of 0.2. The size of the catalog images was 32×32. There were 10 search restarts in the search through the catalog.

Figures 4.13 through 4.19 are organized into seven images/plots per data image: data image, model image, residual image, search trajectory, image profiles, search space top 10%, and search space top 1%. They are described below:

Data, Model, and Residual images: The data image is the image that GMorph attempts to fit a model to. The white line segments in the data image show points that were used to construct the profiles (see below.) The model image is GMorph's best fit to the data image, shown convolved with PSF. The residual image is the model image minus the data image. In the residual image, the image color intensities are rescaled to show detail.

Search trajectory: This shows the different starting points chosen by GMorph, the local minimum that each search arrived at, and the path that each search took. The actual trajectories are in an 8-dimensional parameter space, but are mapped onto the disk radius/bulge radius plane. The mapping is *not* a projection. The disks in the catalog are lined up along the horizontal axis in order of their radii; the dimensions of ellipticity and orientation are ignored. The same is true for the bulges and the vertical axis. Hence two neighbors on an axis may have completely different ellipticities and orientations. Only the first 5 searches are shown in order to simplify the the plots.

Image profiles: This plot shows the profiles of the data image, model image, bulge component of model image, and disk component of model image. The profiles taken from averages of points taken from 7 line segments emanating at 5° increments from the center of the galaxy in the data image. These line segments are shown in white in the data image. Also shown in the lower left of this plot are the final bulge-to-total (B/T), disk radius (r_d), and bulge radius (r_e) that were found by GMorph.

Search space 10% (or 1%): Each combination of bulge and disk in the catalog (here $500 \times 500 = 250,000$ combinations) is evaluated for its match to the data image. Each evaluation is the sum of squared errors determined by a linear regression of the disk and bulge eigencoordinates to the eigencoordinates of the data image. The top 10% (or 1%) of these are plotted as points on this graph and the global minimum within the catalog is marked by a circle. This search space is mapped in the same manner as the search trajectory plots.

The most salient pattern in the results is that the search spaces (the bottom two images) usually have vertical or horizontal 'ridges'. These result when the a disk or bulge is a good enough fit that the properties of the other component are rendered almost irrelevant. Because of this, along most pronounced straight lines the optimal B/T ratio is close to either 0 or 1. Examples of these bulge- or disk-dominated search spaces can be found in Figures 4.13, 4.19, and 4.18.

The search spaces are also marked by many parallel ridges that are unconnected, which seems to indicate many local minima. This is an artifact of the projection. The projection only assures that two neighboring images are the closest (in the catalog) along the r dimension. They may have very different values for ellipticity and orientation. So an apparent valley between two ridges may be created by an image with an identical r parameter, but an ellipticity parameter that spoils the fit completely.

Another pattern in the search trajectories (middle left image) is that there are usually multiple local minima that are found during the search. These local

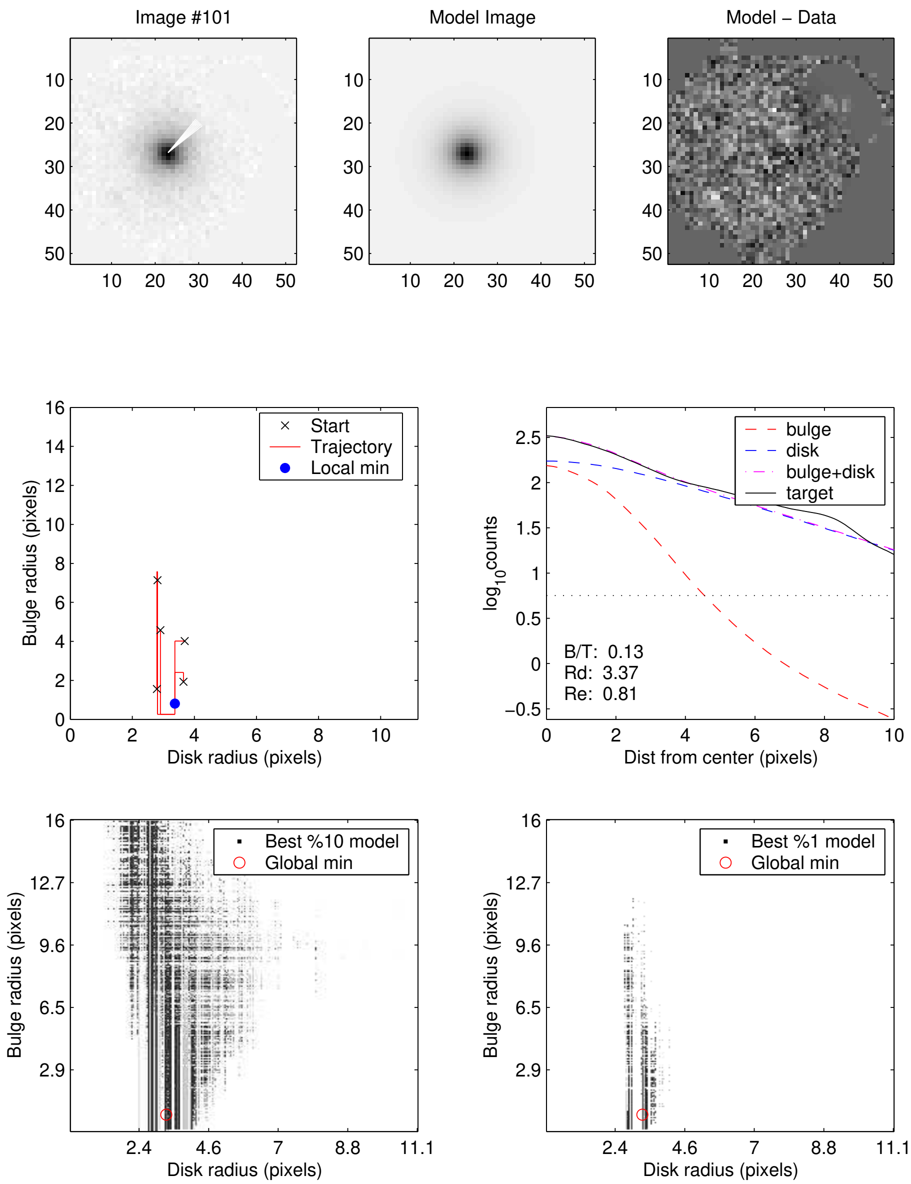


Figure 4.13: Image #101 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

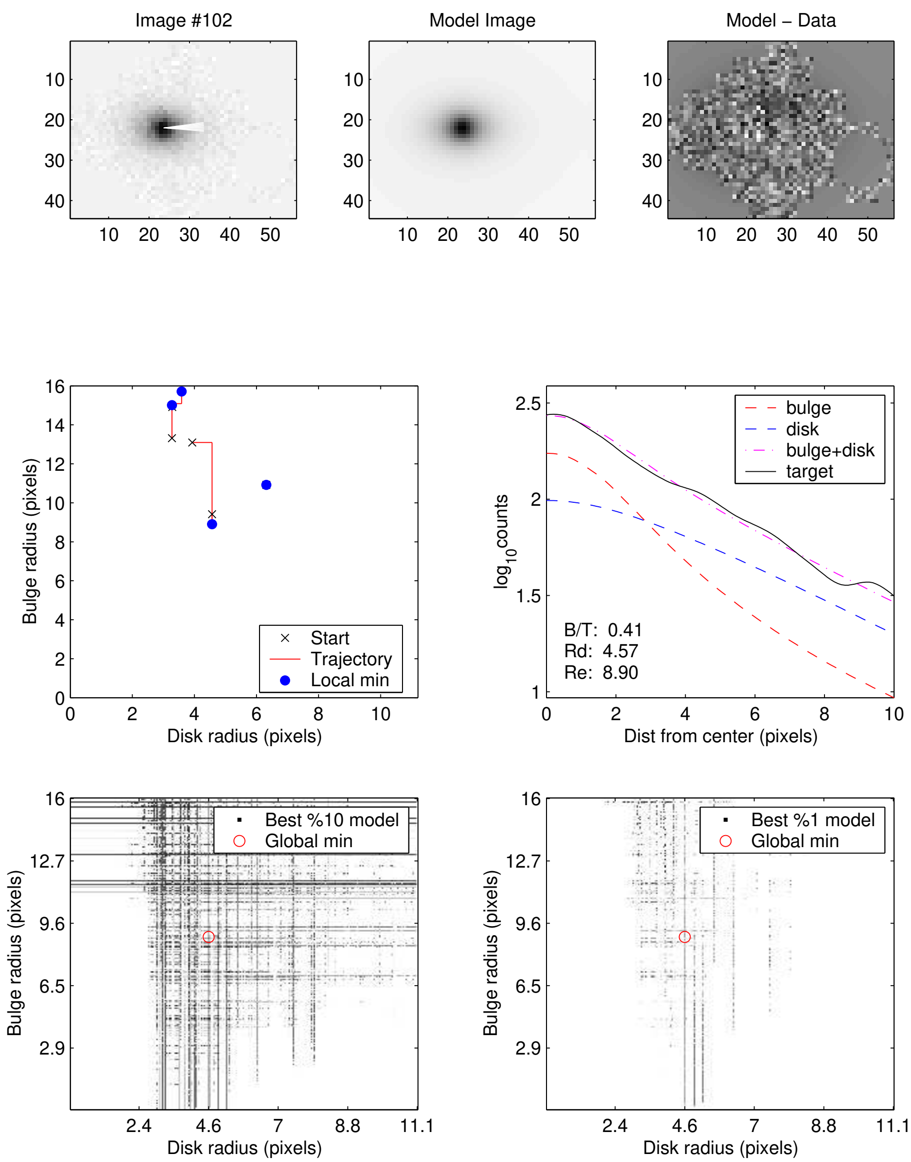


Figure 4.14: Image #102 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

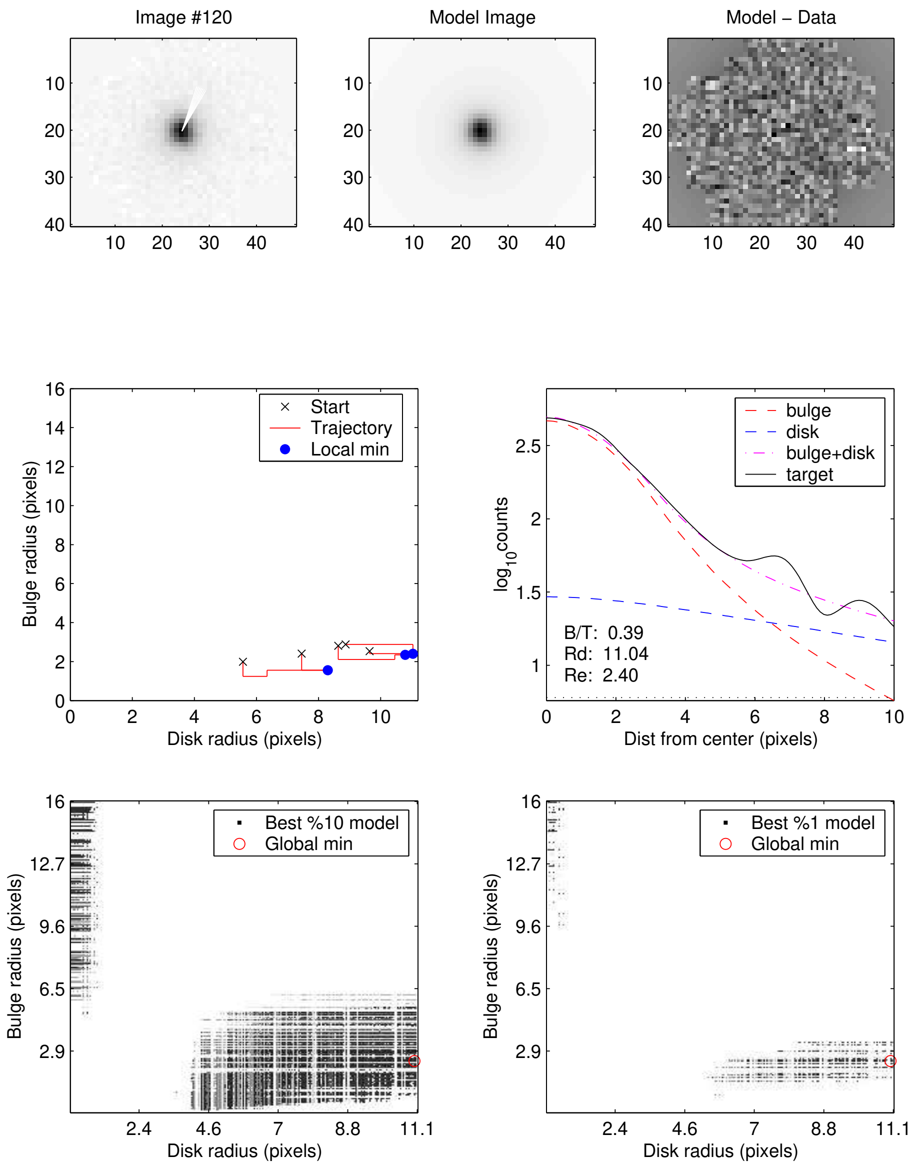


Figure 4.15: Image #120 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

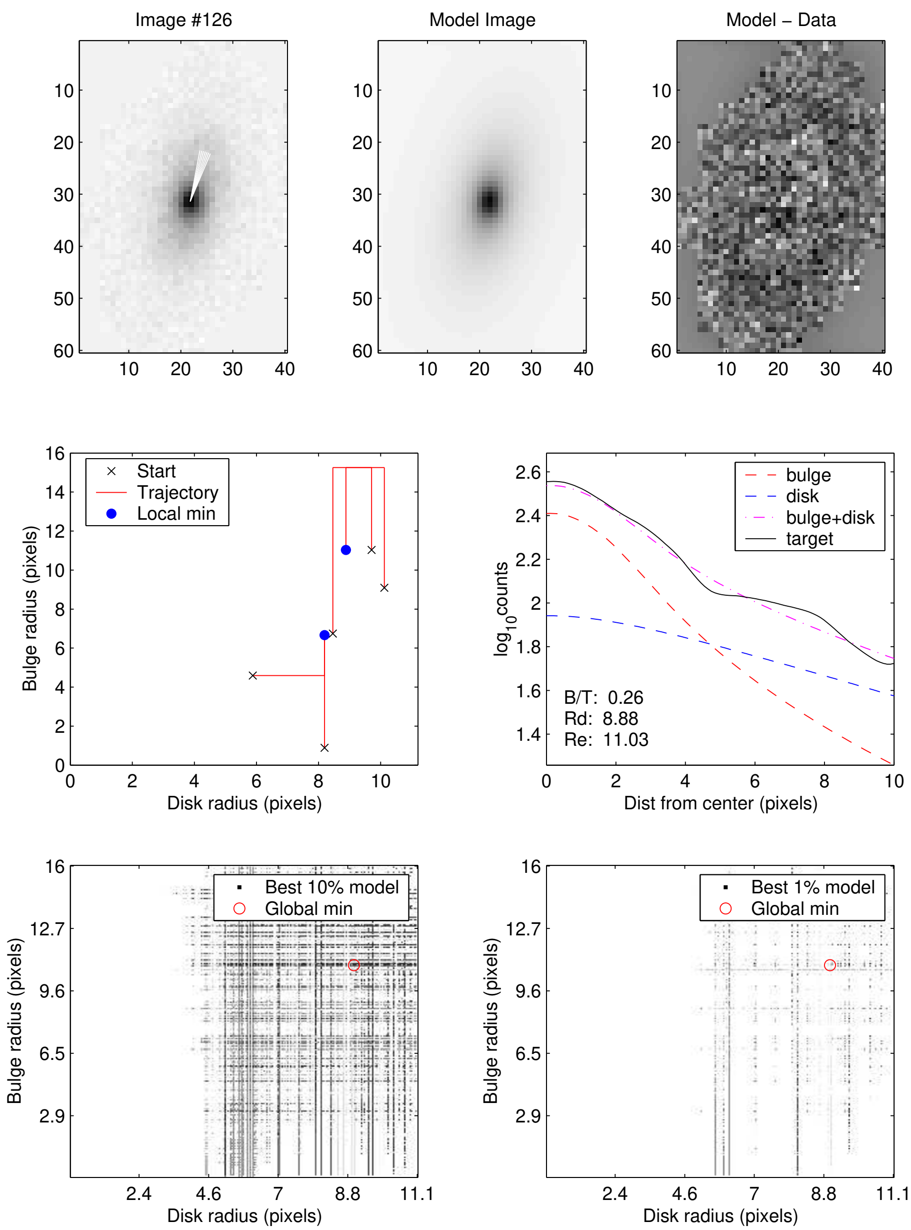


Figure 4.16: Image #126 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

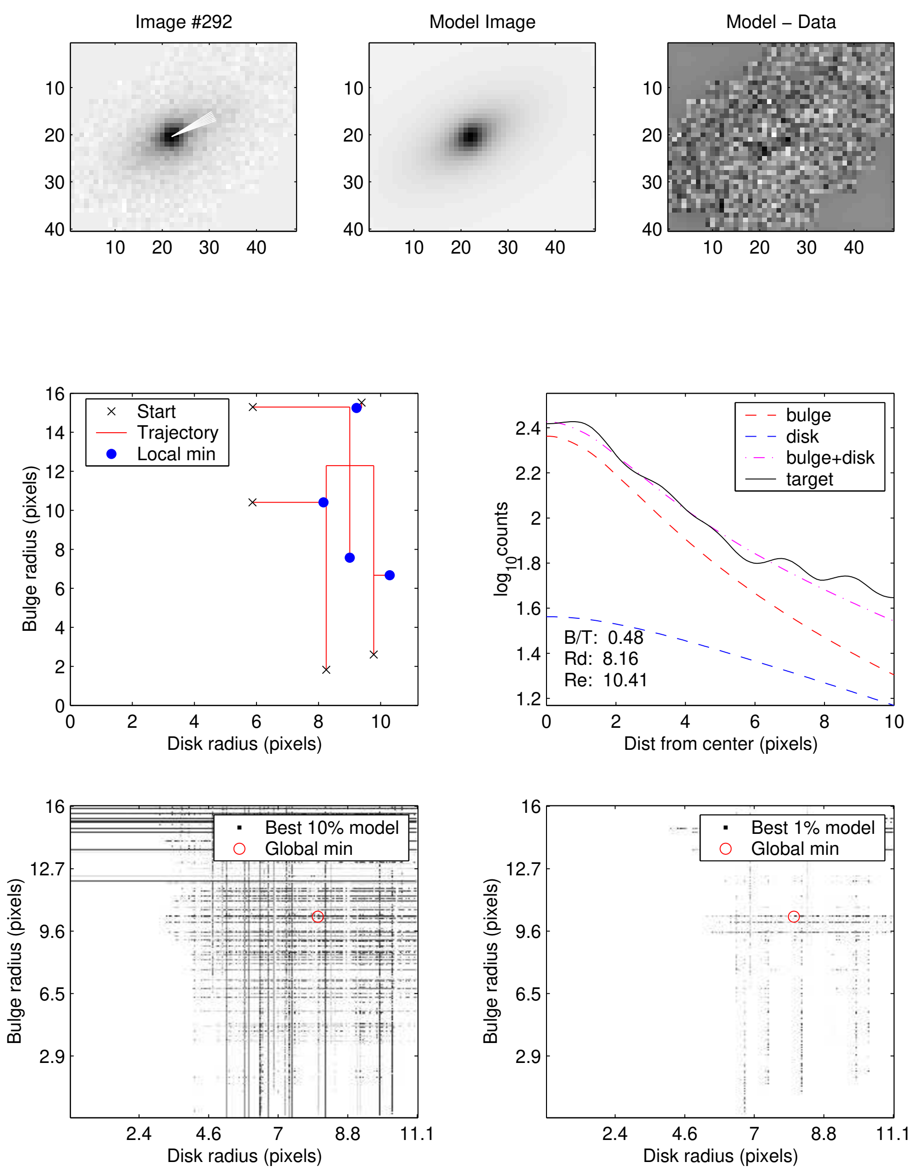


Figure 4.17: Image #292 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

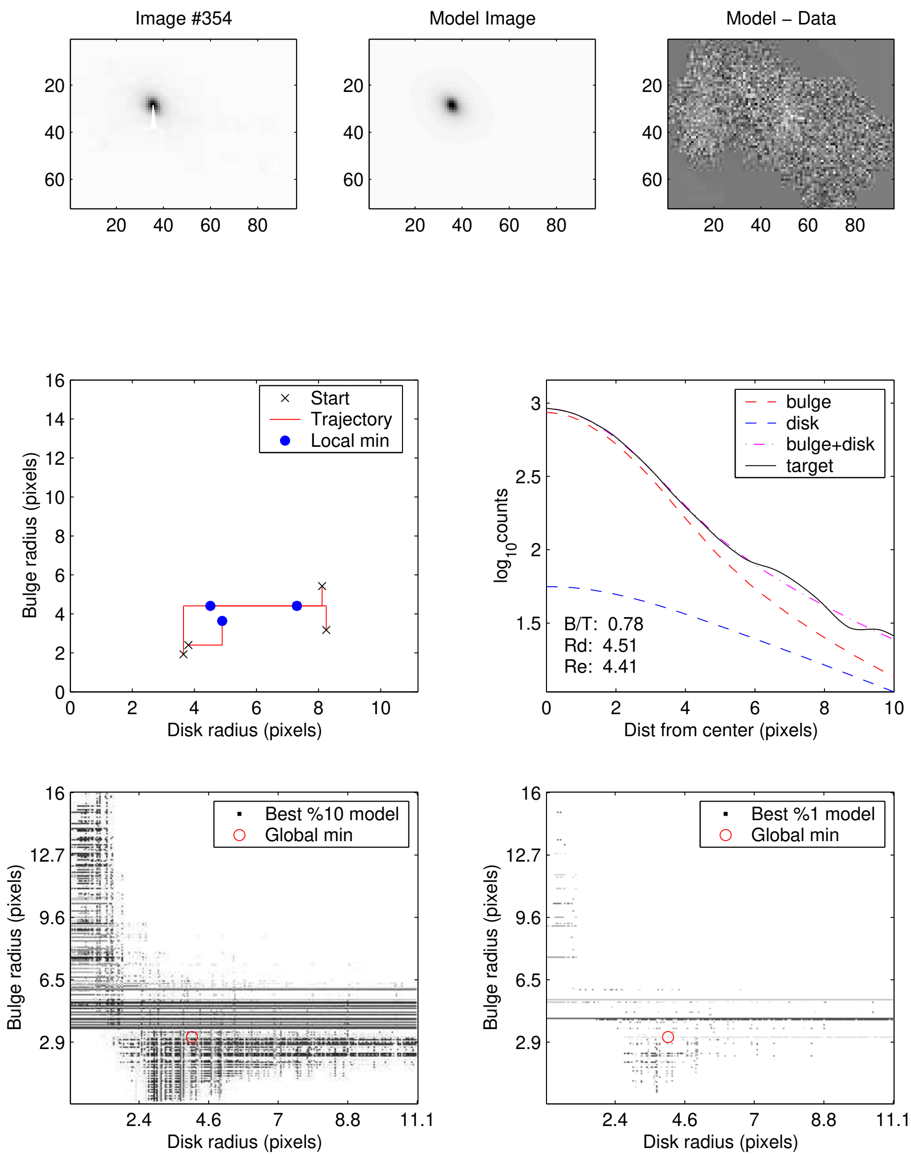


Figure 4.18: Image #354 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

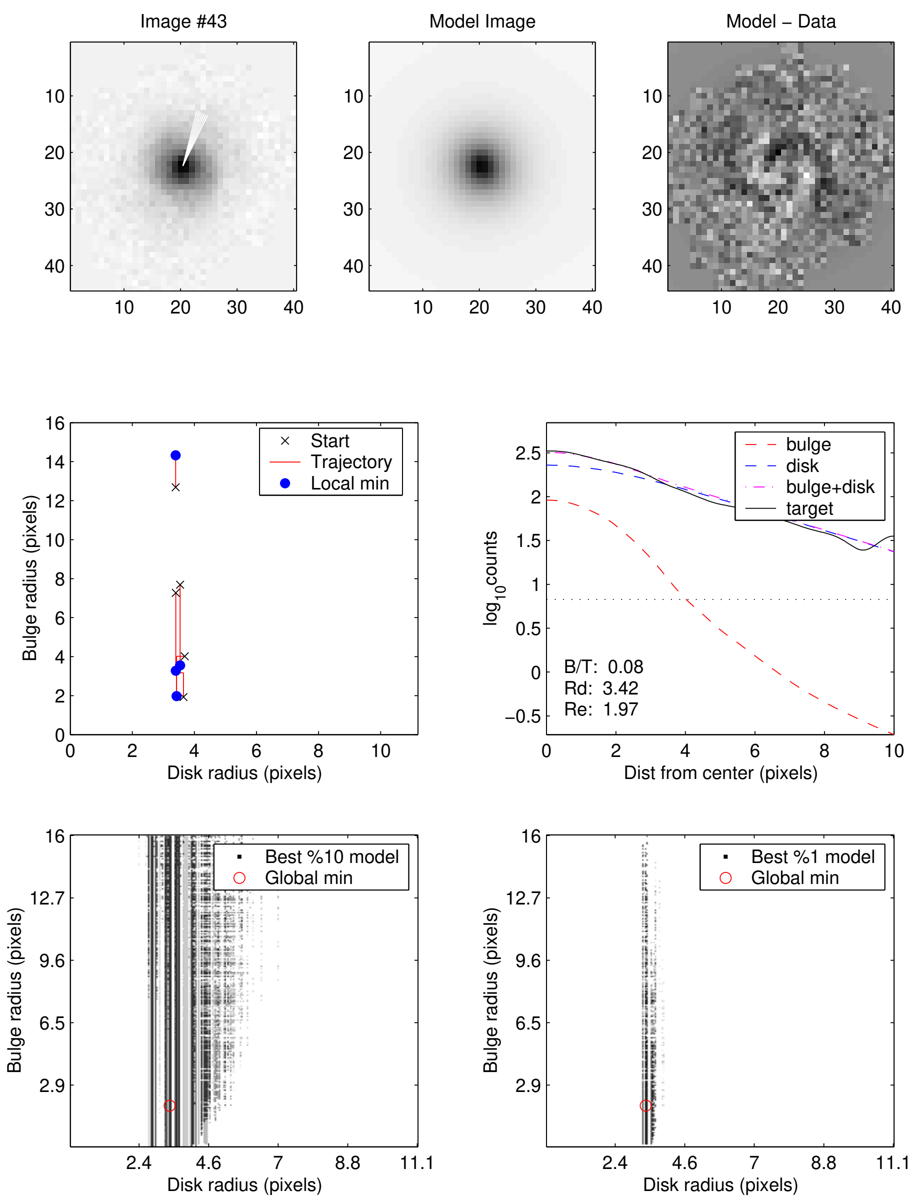


Figure 4.19: Image #43 and resulting fit. The plots, left to right, are: the original image, model image, residual image, search trajectory(ies), data and fitted profiles, and search space with best 10% and 1% of models shown.

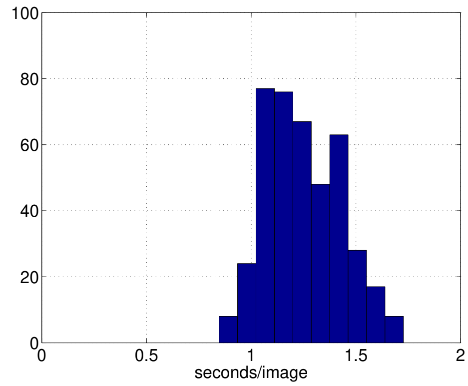


Figure 4.20: Execution times. Distribution of fitting times in seconds for 416 SDSS samples using the procedure described in Section 4.6.1.

minima, however, are only valid for the catalog's discretization and the coordinate descent search method, i.e., another method, such as steepest descent, might be able to continue further. However, the typically wide spread of the local minima suggests that they exist in the continuous search space as well.

The distribution of the execution times for 416 sample images from the SDSS can be found in Figure 4.20. The mean time-to-fit per image was 1.28 seconds. Timings were performed on a 2.8 GHz Pentium 4 desktop computer.

4.6.2 Simulation Experiments Overview

Sets of random galaxy parameters were generated from a uniform distribution in the range $F = 1$, $0 \leq B/T \leq 1$, $-0.5 \leq \mu_x \leq 0.5$, $-0.5 \leq \mu_y \leq 0.5$, $0.25 \leq r_d \leq 16$, $0^\circ \leq \gamma_{inc} \leq 85^\circ$, $0^\circ \leq \gamma_d \leq 180^\circ$, $0.25 \leq r_e \leq 16$,$0 \leq \epsilon \leq 0.7$, and $0^\circ \leq \gamma_b \leq 180^\circ$, and were used to generate 64×64 images of galaxies. These images were then convolved with a Gaussian PSF and had Poisson noise added to them. Some samples of these simulated galaxies are in Figure 4.21.

In the accompanying experiments, unless specified otherwise, the number of eigenimages is 25, the number of pixels per eigenimage is 32×32 (1024), the number of subpixel offsets is 1, and the number of bulge and disk images is 2000 $(1000 + 1000.)$

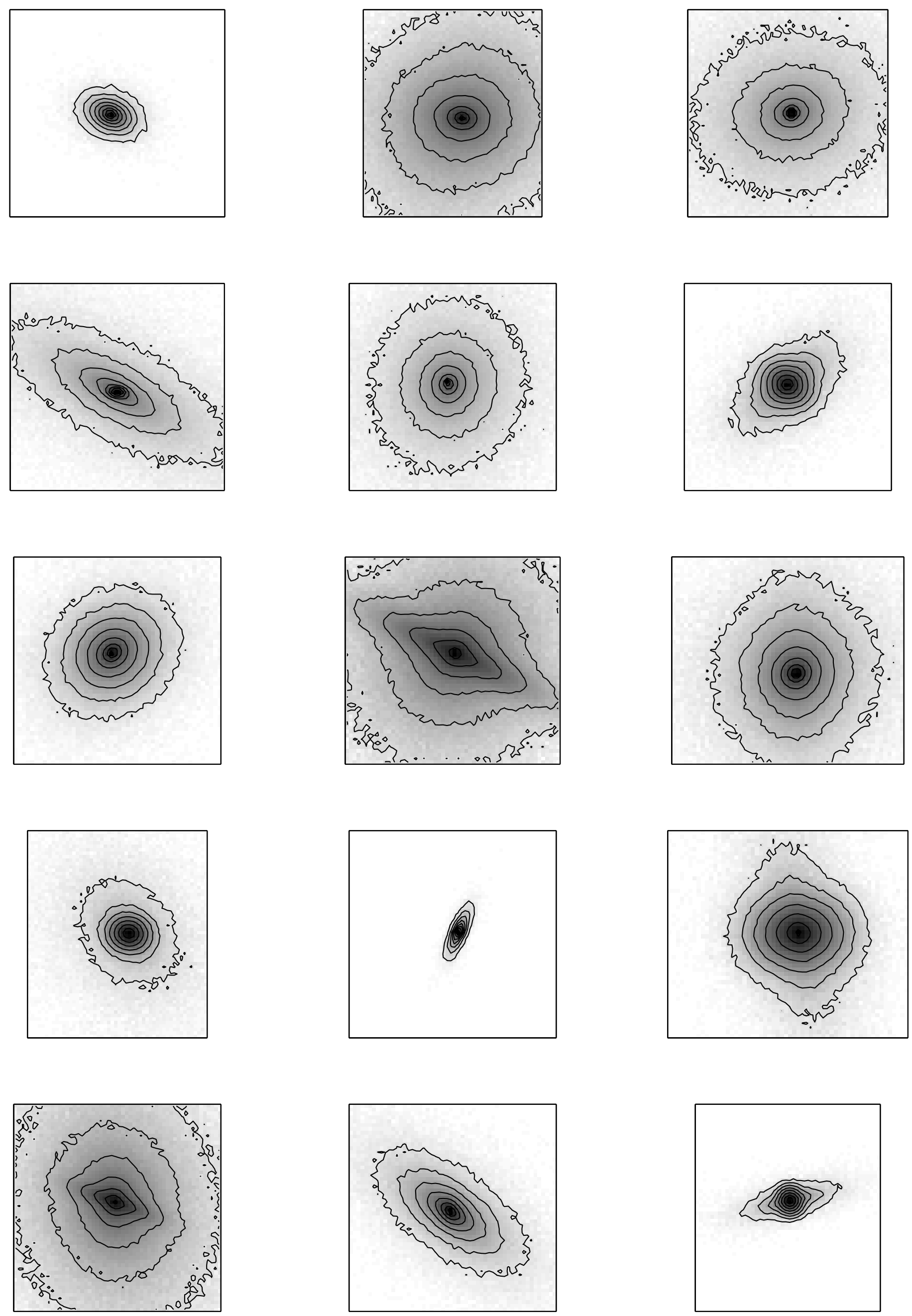


Figure 4.21: Sample of synthetic galaxies. No PSF. Contour lines occur at powers of 10. Signal-to-noise ratio is 100. Background noise is equal to signal noise.

Since images are created from the model itself, there is no model bias effect included in these experiments. That is, no spiral arms, bars, asymmetries, or other unmodelled phenomena are present in the simulated images. Although model bias may have a significant effect on the fitting process in practice, it is excluded for clarity here. How to identify unmodelled structure in an image is a topic for future research.

The source of noise in all cases was Poisson noise that was proportional to the flux at a given pixel. The Poisson probability distribution for a given number of 'counts' c in a pixel (i, j) in the detector is

$$p(c) = \frac{\Psi_{ij}^{c} exp(-\Psi_{ij})}{c!} \tag{4.11}$$

where the flux at that pixel is Ψ_{ij}. The flux is considered to be the expected number of photons or counts at the detector during the exposure. Since the noise is Poisson, both the mean and the variance of c are equal to the value of Ψ_{ij}. The variance for the *total* signal will thus be $\sigma_S^2 = \sum_i^n \sum_j^n \Psi_{ij}$.

Quantification of noise was done using the Signal-to-Noise (S/N) ratio measure. The S/N ratio is defined as

$$S/N = \frac{S}{\sqrt{S+B}} \tag{4.12}$$

where S is the variance of the signal, and B is the variance of the background noise. In these experiments we assume that the background noise is attributable to sky flux, so the background noise will also be a Poisson process.

There is a spectrum of noise regimes that we can model. The two endpoints of this spectrum are signal-dominated and background-dominated noise regimes. The first occurs when the signal of the galaxy is much stronger than that of the background. The effect is to have the noise be heteroskedastic, or of non-uniform variance across the image, since the signal will be non-uniform. The background-dominated regime occurs when the galaxies are very faint and/or the sky is bright. Here the variance is mostly due to the background, so it is uniform across the image and approaches a Gaussian distribution when the counts are greater than about 30. In the following simulations, we model the noise as being a mix of sky and signal. If not otherwise specified, the noise consists of a mix of 50% signal and 50% background.

4.6.3 Timing Experiments

In order to determine the dependence of the execution time on the parameters of the algorithm, a short series of simulation experiments were carried out which some of the algorithm parameters were systematically varied and the effect on speed measured. The parameters in question were the number of eigenimages, the resolution of the eigenimages, the number of subpixel offsets in the catalog, and the number of bulge and disk images in the catalog.

The following rough analysis of the algorithm is meant to explain general patterns in the observed timing results. The time required to fit an image is

$$T_{fit} = \mathcal{O}(KNn_{off} + KC + Kn_{search}(n_b + n_d)) \tag{4.13}$$

The three terms in Equation 4.13 correspond to three stages of the algorithm: the transform of the input image into eigenspace, the sampling of the search space, and the coordinate search of that space (Section 4.5.)

In the accompanying figures, each data point in the plot is the mean fitting time for 1000 simulated galaxies. The error bars cover one standard deviation plus and one minus. The simulated galaxies were created and fit using the procedures in Section 4.6.2. The signal-to-noise ratio was uniformly drawn from the range [100,1000] for each image.

Transform: This involves taking the initial image and finding the best eigenspace and its coordinates in that eigenspace. This stage will roughly take time on order $\mathcal{O}(KNn_{off})$ where K is the dimension of the eigenspace, N is the number of pixels in each eigengalaxy (and the original image), and n_{off} is the number of possible subpixel offsets to attempt to fit. In order to carry

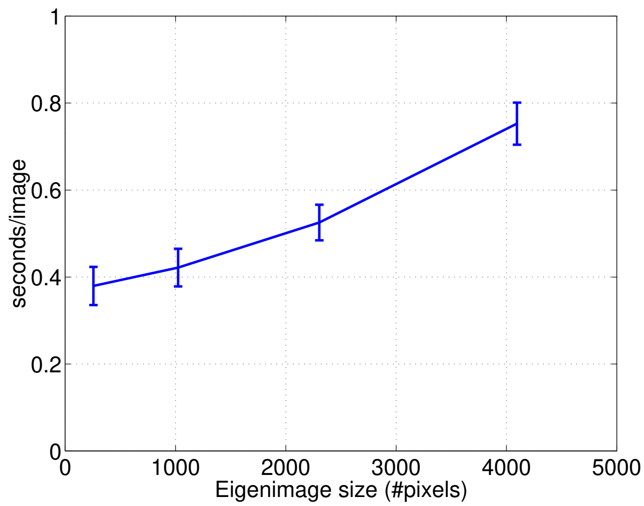


Figure 4.22: Execution times vs. catalog resolution.

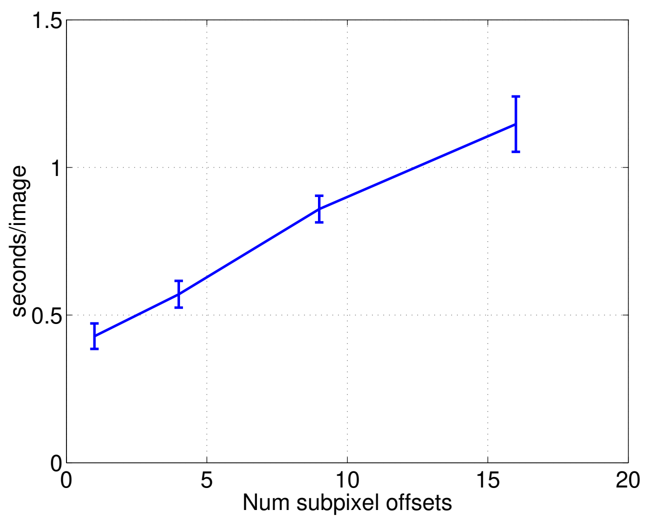


Figure 4.23: Mean execution times vs. number of catalog subpixel offsets.

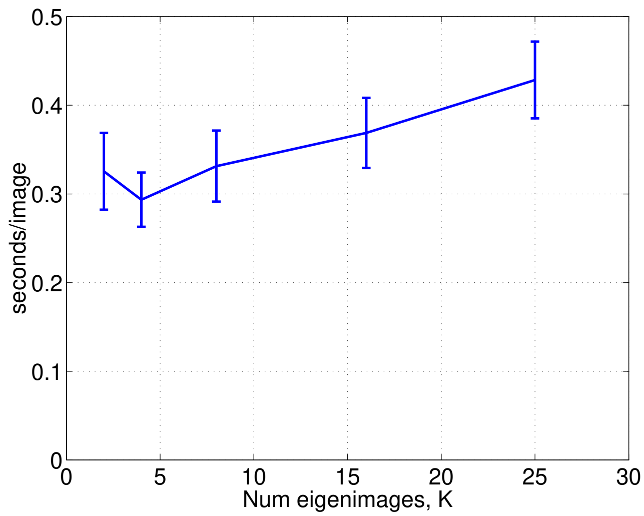


Figure 4.24: Mean execution times vs. number of eigenimages.

out the transform, the original image must be multiplied by each of the K eigenimages (of size N) from each of the n_{off} possible offsets. Figure 4.22 shows how the number of pixels in each eigengalaxy (N) affects the execution time. It appears to be nearly a linear relationship. Figure 4.23 shows how the number of subpixel offsets (n_{off}) increases the execution time in a roughly linear fashion also. The effect of varying the K parameter is displayed in Figure 4.24, where again the relationship appears linear. All of these results are suggested by Equation 4.13.

Sampling: Recall that the search space is first sampled uniformly for good starting points for the search. This stage should take about $\mathcal{O}(KC)$ in the amount of time, where C is the number of samples taken. Each sample requires a linear regression with three dimensions (disk, bulge, and sky) and K data points (one per eigencoordinate.) In all the experiments in this thesis, C was equal to 40,000.

Search: The actual search time should be about $\mathcal{O}(Kn_{search}(n_b + n_d))$, where n_{search} is the number of search restarts and $(n_b + n_d)$ is the number of

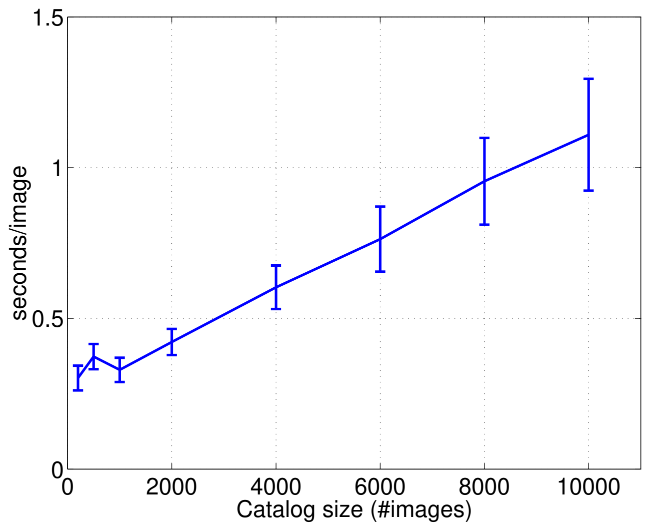


Figure 4.25: Execution times vs. raw catalog size.

bulge and disk images in the catalog. Figure 4.25 shows how the number of catalog images impacts the performance of the algorithm. Again, the relationship is approximately linear.

An interesting observation is the relatively small variance in the search times. GMorph is not gradient-based, so most of the effort of the search is spent during the transform stage. This means that many factors that would affect a gradient-based approach do not significantly affect the execution time of GMorph, e.g., the amount of noise in the image, the quality of the fit, the number of local minima, and the size of the PSF.

Comparing GMorph to other decomposition algorithms shows a clear speed advantage. GIM2d running 155 50×50 images on the same computer took 9.3 hours, giving an average running time of 216 seconds per image. Another algorithm, Galfit, based on Levenberg-Marquardt, is a fast gradient-based optimization routine. Galfit reports an approximate 20-fold speedup over GIM2d, putting execution time at approximately 10 seconds per 50×50 image [40]. As the previous figures have shown, the equivalent execution time for GMorph lies somewhere between 0.5 and 1.5 seconds, depending on the parameters of the catalog used.

4.6.4 Catalog Parameter Experiments

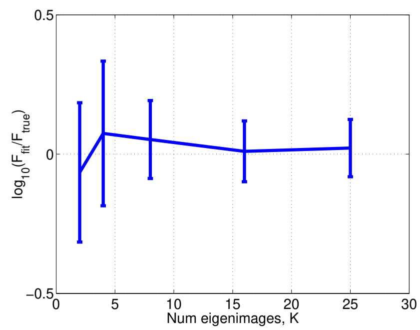
Besides speed, the catalog parameters of size, resolution, and number of offsets have other effects, namely, the accuracy of the algorithm. To measure accuracy, we use as a surrogate the error in estimating the total flux of the image.

In the accompanying figures, unless specified otherwise, the number of eigenimages is 25, the number of pixels per eigenimage is 32×32 (1024), the number of subpixel offsets is 1, and the number of bulge and disk images is $1000 + 1000$. The signal-to-noise ratio was uniformly drawn from the range [100,1000] for each image. Each data point in the plot is the mean fitting time for 1000 simulated galaxies. The error bars cover one standard deviation plus and one minus. The simulated galaxies were created and fit using the procedures in Section 4.6.2.

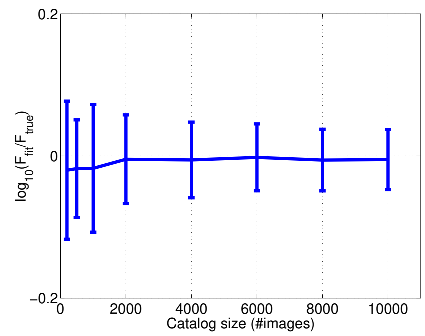
Figures 4.26a and 4.27a show the effect on accuracy of increasing the number of eigenimages. The initial effect is to increase accuracy, but the advantage in adding eigenimages beyond 8-16 diminishes rapidly. These results were also suggested in Figure 4.6, which showed the percentage of variance explained by different numbers of eigenimages. Both results suggest the use of about 8-16 eigenimages for good accuracy without an undue cost in speed.

Figures 4.26b and 4.27b present evidence that increasing the number of images in the catalog reduces the error in the fitted flux parameter. The majority of this benefit is achieved by a catalog size of 3000 bulge and 3000 disk images.

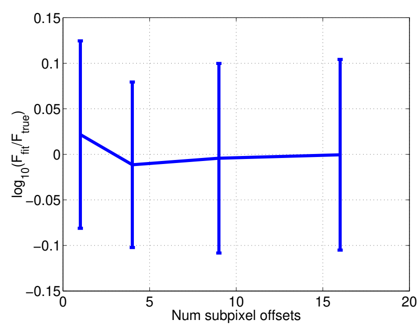
Figure 4.26c shows the effect of increasing the number of subpixel offsets. The measured bias in the fits is mostly eliminated by about 16 offsets (i.e., a 4×4 grid per pixel.) On the other hand, Figure 4.27c indicates that little change in the mean absolute error occurs as the number of offsets is increased. Neither result is applicable in general, however, since the relationship between the size of the features of galaxies and the resolution of the eigenimages determines the necessity of subpixel offsets. In other words, the size of the pixels compared to



(a)

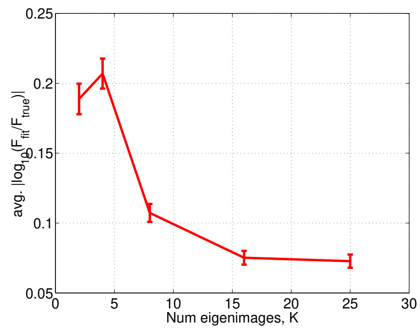


(b)

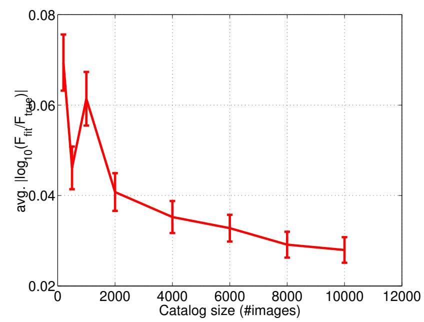


(c)

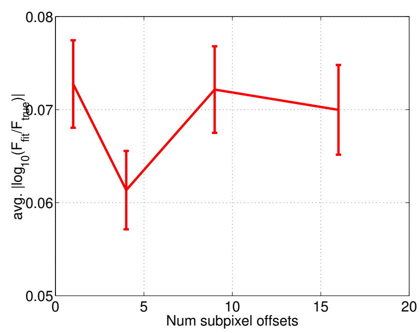
Figure 4.26: The effect of the number of eigenimages, the number of catalog images, and the number of subpixel offsets on flux accuracy. Each data point is the mean result for 1000 simulated galaxies. Vertical bars cover one standard deviation in each direction.



(a)



(b)



(c)

Figure 4.27: The effect of various algorithm parameters on flux accuracy. Plots are of the mean absolute log(fitted/true) of the flux values. The vertical bars cover a 95% confidence interval for the estimated mean. Each data point summarizes the results for 1000 simulated galaxies.

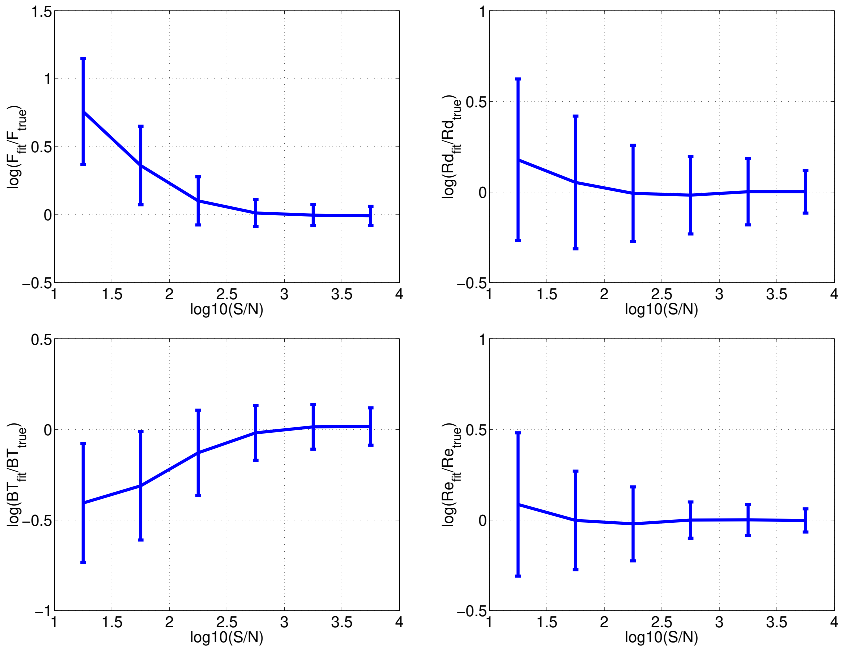


Figure 4.28: Fitting errors with respect to S/N. Error is the log(fitted/true) of the parameter values. Each point is the mean of a minimum of 150 different error measurements, and the vertical bars cover one standard deviation in each direction.

the size of the galaxies will determine the optimal number of subpixel offsets.

4.6.5 Noise and PSF Experiments

These experiments look at two factors that tend to affect the accuracy of fits the most: noise and blur. Both factors degrade the image, and can introduce spurious signal. The effect of these factors were measured by looking at their effect on four of the twelve parameters: total flux (bulge + disk flux), bulge-to-total flux ratio, disk radius, and bulge radius. The effect was quantified as the $\log_{10}$ of the ratio between the actual parameters and the values fitted. In order to test this, at least 150 simulated galaxies were generated per cell, and GMorph determined a fit for each. Figures 4.28 and 4.29 show how well these parameters are fit under different signal-to-noise conditions. These measurements assume that the PSF is accurate. They do not address the possibility of

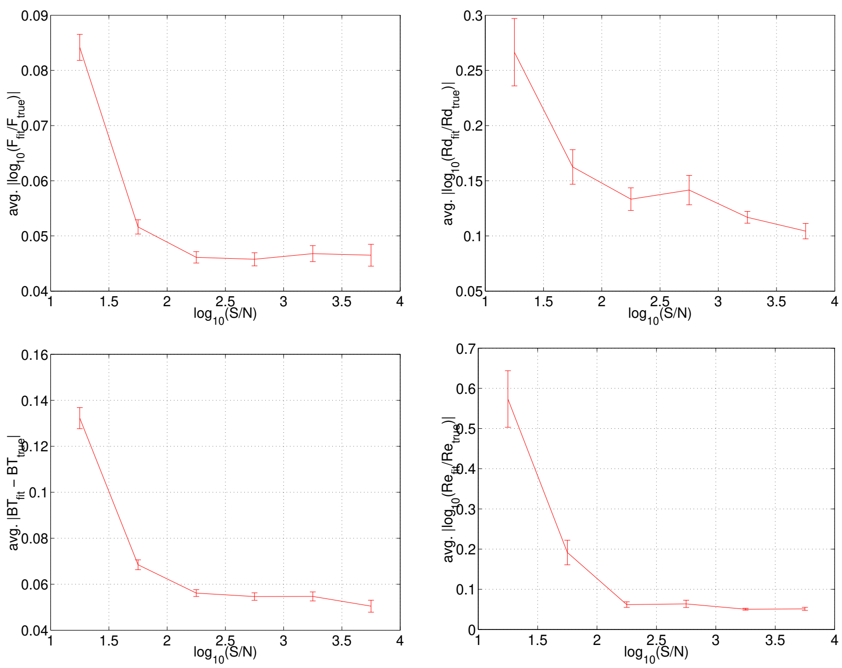


Figure 4.29: Magnitude of fitting errors with respect to S/N. Plots are of the average absolute log(fitted/true) of the parameter values. The vertical bars cover a 95% confidence interval for the estimated mean. Note that B/T error is the average absolute difference.

errors introduced by PSF mismatch.

Generally speaking, the low-signal condition produces larger systematic errors than increasing the size of the PSF (over the noise and size ranges used here.) As one would hope, the systematic error disappears as the strength of the signal is increased.

In Figure 4.28, both the total flux and B/T parameters have the greatest systematic effects resulting from noise. The cause of both of these deviations is explained by the disk flux being a too large and the bulge flux being too small (by a lesser amount) in noisy conditions. This results in a too-large total flux (bulge flux + disk flux) and a too-small B/T ratio (bulge flux / (bulge + disk flux)).

Tables 4.1, 4.3, 4.4, and 4.2 summarize in greater detail the systematic errors and random errors that occur in various conditions of noise and blurring. For

Signal-to-Noise	PSF σ (pixels): 0	1	2	4	8
$10^1 - 10^{1.5}$	0.759	0.943	1.030	1.071	1.232
	0.362	0.452	0.537	0.701	0.823
$10^2 - 10^{2.5}$	0.102	0.164	0.241	0.389	0.542
	0.013	0.032	0.057	0.142	0.319
$10^3 - 10^{3.5}$	-0.003	-0.001	-0.005	0.025	0.123
	-0.008	-0.009	-0.008	-0.005	0.026

Signal-to-Noise	PSF σ (pixels): 0	1	2	4	8
$10^1 - 10^{1.5}$	*0.391*	*0.327*	*0.292*	*0.353*	*0.338*
	0.289	*0.289*	*0.281*	*0.250*	*0.292*
$10^2 - 10^{2.5}$	*0.177*	*0.194*	*0.213*	*0.228*	*0.253*
	0.100	*0.112*	*0.131*	*0.182*	*0.222*
$10^3 - 10^{3.5}$	*0.078*	*0.063*	*0.074*	*0.097*	*0.181*
	0.070	*0.057*	*0.059*	*0.068*	*0.115*

Table 4.1: Systematic and random errors in total flux. Upper table: Mean difference between $\log(F)$ and recovered $\log(F)$ as a function signal-to-noise and PSF size. Lower table: Standard deviation in difference between $\log(F)$ and recovered $\log(F)$.

example, at high S/N, the PSF size does not have a strong systematic effect, however Tables 4.1 through 4.2 show a strong interaction effect between low S/N and PSF size. The interaction amplifies the direction of the deviation introduced by the PSFs in the high S/N conditions.

They are only one slice through the complex relationships among the parameters, however. The systematic errors also vary according to the bulge ellipticity, B/T, r_e, and r_d of the input image, for instance. For example, in low S/N, the bulge radius and the disk radius are negatively correlated. Some of these relationships can be found in the correlation matrices in Tables 4.5 and 4.6.

Occasionally, at low signal-to-noise, the algorithm makes the radius of the disk (or bulge) as small as is allowed (10^{-12} pixels). This is clearly not physically probable, and results from too many degrees of freedom and not enough signal, i.e., over-fitting. A likely overfitting scenario would be the algorithm deciding to fit the central high-value pixel perfectly, and use the rest of the model (bulge and sky) to explain the remainder of the image. The algorithm/model currently has no *a priori* constraints on what is unphysical; the microscopic disks and bulges

Signal-to-Noise	0	1	2	4	8
$10^1 - 10^{1.5}$	-0.406	-0.436	-0.435	-0.400	-0.331
	-0.311	-0.377	-0.378	-0.375	-0.369
$10^2 - 10^{2.5}$	-0.129	-0.197	-0.226	-0.339	-0.324
	-0.019	-0.040	-0.093	-0.174	-0.257
$10^3 - 10^{3.5}$	0.014	0.009	-0.008	-0.041	-0.162
	0.016	0.017	0.015	0.014	-0.029

PSF σ (pixels)

Signal-to-Noise	0	1	2	4	8
$10^1 - 10^{1.5}$	*0.327*	*0.323*	*0.318*	*0.354*	*0.425*
	0.299	*0.311*	*0.301*	*0.320*	*0.368*
$10^2 - 10^{2.5}$	*0.235*	*0.266*	*0.283*	*0.299*	*0.326*
	0.151	*0.163*	*0.215*	*0.270*	*0.312*
$10^3 - 10^{3.5}$	*0.123*	*0.103*	*0.122*	*0.179*	*0.276*
	0.103	*0.087*	*0.102*	*0.109*	*0.225*

PSF σ (pixels)

Table 4.2: Systematic and random errors in B/T ratio. Upper table: Mean difference between B/T and recovered B/T as a function signal-to-noise and PSF size. Lower table: Standard deviation in difference between B/T and recovered B/T.

Signal-to-Noise	0	1	2	4	8
$10^1 - 10^{1.5}$	0.178	0.265	0.338	0.391	0.506
	0.053	0.087	0.184	0.287	0.429
$10^2 - 10^{2.5}$	-0.007	0.015	0.055	0.164	0.353
	-0.017	-0.002	0.005	0.072	0.205
$10^3 - 10^{3.5}$	0.002	-0.010	0.002	0.011	0.088
	0.002	0.002	-0.003	0.024	0.048

PSF σ (pixels)

Signal-to-Noise	0	1	2	4	8
$10^1 - 10^{1.5}$	*0.446*	*0.433*	*0.451*	*0.432*	*0.400*
	0.366	*0.366*	*0.394*	*0.423*	*0.411*
$10^2 - 10^{2.5}$	*0.265*	*0.254*	*0.283*	*0.403*	*0.410*
	0.214	*0.187*	*0.201*	*0.262*	*0.419*
$10^3 - 10^{3.5}$	*0.183*	*0.152*	*0.162*	*0.192*	*0.317*
	0.118	*0.089*	*0.067*	*0.151*	*0.191*

PSF σ (pixels)

Table 4.3: Systematic and random errors in disk radius . Upper table: Mean difference between $\log(r_d)$ and recovered $\log(r_d)$ as a function signal-to-noise and PSF size. Lower table: Standard deviation in difference between $\log(r_d)$ and recovered $\log(r_d)$.

Signal-to-Noise		PSF σ = 0	1	2	4	8
	$10^1 - 10^{1.5}$	0.086	0.188	0.275	0.283	0.350
		-0.002	0.059	0.142	0.257	0.306
	$10^2 - 10^{2.5}$	-0.021	-0.026	0.017	0.114	0.168
		0.000	-0.001	-0.031	0.000	0.068
	$10^3 - 10^{3.5}$	0.001	0.001	-0.008	-0.006	0.056
		-0.002	-0.006	-0.009	-0.018	0.003

PSF σ (pixels)

Signal-to-Noise		PSF σ = 0	1	2	4	8
	$10^1 - 10^{1.5}$	*0.395*	*0.438*	*0.482*	*0.536*	*0.514*
		0.272	*0.348*	*0.403*	*0.447*	*0.463*
	$10^2 - 10^{2.5}$	*0.204*	*0.247*	*0.313*	*0.377*	*0.458*
		0.100	*0.142*	*0.209*	*0.320*	*0.428*
	$10^3 - 10^{3.5}$	*0.085*	*0.083*	*0.113*	*0.201*	*0.369*
		0.064	*0.062*	*0.072*	*0.106*	*0.247*

PSF σ (pixels)

Table 4.4: Systematic and random errors in bulge radius. Upper table: Mean difference between $\log(r_e)$ and recovered $\log(r_e)$ as a function signal-to-noise and PSF size. Lower table: Standard deviation in difference between $\log(r_e)$ and recovered $\log(r_e)$.

B/T	1.00									
r_d	-0.11	1.00								
γ_{inc}	0.02	0.02	1.00							
r_e	-0.07	-0.03	-0.00	1.00						
ϵ	-0.15	-0.09	0.05	0.10	1.00					
F error	-0.11	0.09	0.18	0.04	-0.04	1.00				
B/T error	0.17	-0.10	-0.09	0.06	0.06	0.00	1.00			
sky error	0.05	0.06	-0.01	-0.01	-0.09	0.86	0.02	1.00		
r_d log error	-0.13	0.17	0.08	0.04	-0.05	0.05	-0.05	0.03	1.00	
r_e log error	0.08	-0.01	-0.00	0.11	0.11	-0.01	0.10	-0.02	0.00	1.00

Table 4.5: Correlations in high-signal conditions. Correlations between selected parameters and types of errors when the total signal-to-noise is greater than 1000. Sample of 3000 simulated galaxies with no PSF.

B/T	1.00									
r_d	-0.36	1.00								
γ_{inc}	-0.24	-0.02	1.00							
r_e	0.33	-0.24	-0.01	1.00						
ϵ	0.40	-0.27	-0.07	0.64	1.00					
F error	-0.07	0.31	0.08	0.08	0.09	1.00				
B/T error	0.44	-0.19	-0.17	0.27	0.17	0.02	1.00			
sky error	0.34	-0.14	-0.19	0.22	0.31	0.67	0.18	1.00		
r_d log error	-0.21	0.44	0.12	-0.07	-0.09	0.11	-0.15	-0.12	1.00	
r_e log error	0.49	-0.27	-0.08	0.62	0.94	0.15	0.19	0.40	-0.11	1.00

Table 4.6: Correlations in low-signal, high-noise conditions. Correlations between selected parameters and types of errors when 10<S/N<100. Sample of 3000 simulated galaxies with no PSF.

are flagged as being questionable data points. In the reported experiments this occurred during about 5% of the fits, and have been excluded from the experimental results. It is a subject of future research to determine the proper constraints to place on GMorph's search space.

4.7 Summary

This section has described an algorithm that can significantly speed up the automatic identification of galaxies. Memory-based methods allow for fast identification of these galaxies through sampling of the parameter space, the use of eigenimages, and decomposing the image model. In order to deal with the blurring induced by PSFs, we use a combination of pre-blurred local eigenspaces and correction of misfits by linear regression. We have measured the performance of the algorithm under varying noise and PSF conditions via simulation and mapped out some of the systematic and random errors occurring in the fitting process. Some of the correlations in the errors are also shown. The data suggest a range of signal-to-noise values that are safe for a given tolerance for error in the parameter of interest. The GMorph method can accurately determine the 2-dimensional morphology of galaxies in a fraction of the time required by current 2-dimensional techniques. GMorph could also be used in conjunction with other techniques, e.g., Quasi-Newton descent.

Chapter 5

Pairwise Bisection Optimization

This chapter[1] introduces a new nonparametric algorithm designed to optimize stochastic functions. Robustness to noise and outliers and use of all data are the main advantages of this method. We will first describe the algorithm, then test its performance against noise-hardened Nelder-Mead simplex search, PMAX, and Q2. PABI uses nonparametric statistics to adjust for noise. This algorithm chooses an experiment by iteratively shrinking a region of interest (ROI) around a suspected optimum using a novel application of the nonparametric statistic, Kendall's tau [22]. During this process, the algorithm must decide when to stop shrinking, which equates to a decision about bias-variance tradeoff [9]. In a perfect world, one would minimize variance by keeping a large set of points, thereby ensuring a good fit of your model. However, one must also minimize bias by sampling from a small area, thereby avoiding force-fitting an overly simple model. In the algorithm, this decision is made using the same statistic, Kendall's tau.

PABI is derived from the Q2 model of optimizers (see Section 2.3.4), but

[1]Much of the material presented in this chapter has been published in [2] with co-authors Andrew Moore and David Cohn.

with an important difference: it does not use polynomials to model the function. Instead, the algorithm models the function nonparametrically. This produces some major benefits: 1) robustness to non-Gaussian noise and outliers, 2) reduced data consumption, 3) ability to operate in hybrid continuous-discrete spaces, and 4) no assumption of smoothness of underlying function. Empirical has shown that the approach has superior performance to Q2 in several test domains.

5.1 Background

Because the function to be optimized will typically be unknown before optimization takes place, we wish to create a guiding model of minima which has simple and hopefully robust assumptions about the shape of the minimum. Therefore, we restrict ourselves to the following assumption: the outputs of points closer to the minimum are smaller than those of points farther away. Given a set of points and a candidate minimum x^* we assume that, locally

$$f(x) = g(\|x - x^*\|) \qquad g\, monotonic \tag{5.1}$$

Since we do not specify the shape of the function g, the only parameter for a model k is x_k^*. Having few model parameters allows efficient use of the data to make inferences, thus requiring fewer experiments. The shape of the minimum can take any form so long as the function is monotonically increasing with respect to the distance from the minimum [20]. Although this imposes a strong assumption of symmetry with respect to the distance metric $\|\cdot\|$, there is no requirement on $f(x)$ for continuousness or any prespecified noise distribution for $y(x)$. Another useful property of this model is immunity to all monotonic transformations of the data [42].

Observed data will fit this model of minima to varying degrees, so one would like to quantify the degree of fit. Given the assumption of Equation **??**, we will use a nonparametric method to test the positive monotonic association between

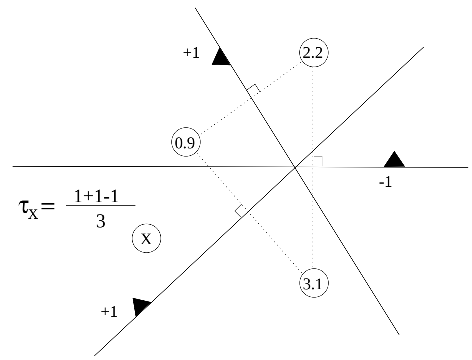


Figure 5.1: Example calculation of the statistic τ with two input dimensions. Each circle is a point in two-dimensional space, with the y value inside the circle. Solid lines are the perpendicular bisectors of each pair of points. For each pair's bisector, a solid triangle indicates the direction of the smaller of the two points. A $+1$ or -1 indicates whether the query point X is on the same side of the bisector as the smaller of the two points. The statistic τ_X is equivalent to the sum of these numbers divided by the number of bisectors.

$\|x - x^*\|$ and $f(x)$. A typical nonparametric measure used for this purpose is *Kendall's correlation coefficient*, otherwise known as *Kendall's tau* [22]. We will denote the Kendall's tau for the relationship between $\|x - x^*\|$ and $y(x)$ in the data set D as $\tau_{x^*,D}$. In this chapter, the subscript D will be omitted whenever possible.

An intuitive geometric interpretation of the appearance of Kendall's tau here is the following: if one creates a perpendicular bisector between a pair of points x_1 and x_2, our model of a minimum will be satisfied only if the minimum is on the same side of the bisector as the point with the smaller $y(x)$. One can imagine creating n *choose* 2 bisectors in space, one for each pair of points in the data set. The number of satisfied bisectors minus the number of unsatisfied bisectors, divided by the total number of bisectors, will have the same distribution as Kendall's tau.

More generally, we wish to calculate τ_m for the model of Equation **??** with $x^* = m$ and data set D. To do this, we acquire the paired data, $\|x_i - m\|$ and y_i, and denote their ranks by r_i and s_i respectively. Next, arrange the r_i in ascending order (so that $r_i = i$.) A monotonic association between $\|x - m\|$ and

$y(x)$ should cause the s_i to show either an increasing or decreasing trend. To measure this, we score each paired difference $s_j - s_i$ for $i = 1, 2, ..., n-1$ and $j > i$ as $+1$ if the difference is positive and as -1 if negative. Kendall called a positive difference a *concordance* and a negative difference a *discordance*. Denoting the sums of concordances and discordances by n_c and n_d respectively, the equation for our tau is

$$\tau_m = \frac{n_c - n_d}{\frac{1}{2}n(n-1)} \tag{5.2}$$

If the monotonic association is perfectly positive, $\tau_m = 1$, and perfectly negative association gives $\tau_m = -1$. If the rankings of $\|x - m\|$ and y are independent, we expect τ_m to be close to 0. Ties in the rankings pose no special difficulty.

5.1.1 PABI

The Pairwise Bisection (PB1) algorithm [2] has two major phases. The first phase is *sample reduction*, in which the data set is narrowed to a subset of promising points. The second phase is *experiment selection*, which consists of choosing a point x from the area defined by the reduced sample generated in the first phase.

Selecting a Region of Interest

The purpose of sample reduction is to select a promising subset of the known data points. The reduced sample will ideally possess the following properties:

A. Covers a small enough section of the input space to eliminate the bias of the monotonic model (Equation 5.1)

B. Contains enough points to counter the effects of noise

C. Convexity

Not all these conditions can usually be satisfied at once, so sample reduction makes tradeoffs among them. The sample reduction phase used by PB1 consists of finding the reduced set of points, D_{red}. At the global level, the monotonic-minimum model is almost certainly biased. However, we will assume that it is still useful as a heuristic indicator of unpromising points. These unpromising points are gradually eliminated until a small set of points remains.

The sample reduction phase consists of the following three steps :

1. Include all n known points into sample set D_0.

2. For every point j in D_i, calculate τ_j. Generate D_{i+1} by removing the exterior [2] point in D_i with the smallest τ. Repeat this step for D_0 through D_{n-1}.

3. Decide which of these sample sets, D_0 through D_{n-1}, to use by choosing the *smallest sample set with a significant* τ. The determination of significance in this case is decided by applying a *two-tailed* hypothesis test to all points in D_i (i.e., one of the points' τ must be either significantly high or low.)

Intuitively, since the τ measure detects basin-like structures, steps 1 and 2 will tend to produce reduced samples that converge on local minima. When this is the case, always selecting the smallest reduced sample in step 3 will favor condition **A**, and the requirement of structural significance will enforce condition **B**. The final condition **C** is directly ensured by step 2, because points are only allowed to be removed from the "exterior" of the sample.

Selecting an Experiment

The purpose of this phase is to select a point inside the region defined by the points in the reduced sample. The experiment selection phase consists of two steps:

1. Define the Region of Interest (ROI). The *ROI* is defined as the set of all points in the input space which meet the criteria given by Equations 5.3

[2] An exterior point is defined here to be any point which is furthest from the centroid of the sample when all points are projected onto the line through it and the centroid. This is intended to be an approximation of the exterior points of the minimum-area convex polyhedron.

and **??**. For all points x in ROI,

$$\min_{y \in D_{red}} \|x - y\| \leq \min_{z \in (D - D_{red})} \|x - z\| \tag{5.3}$$

This describes a Voronoi manifold from within which the experiment must be drawn. The other requirement is that all points in the ROI must also fit the data in the reduced sample, i.e., for all x in ROI,

$$\tau_{x,D_{red}} \geq \tau^{\alpha} \tag{5.4}$$

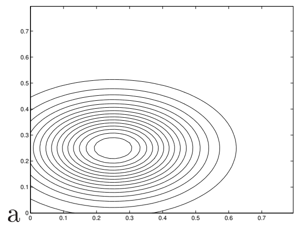
Where τ^{α} is a critical value for τ that indicates a significance level of α (typically 0.05). This value only depends on the number of points in D_{red}, and is precalculated.

2. Choose g experiments uniformly randomly from within ROI. From these g points, with probability p choose the point with the best τ. With probability $(1 - p)$ choose the point furthest away from all previously seen points.

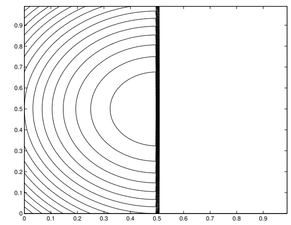
The above procedure is currently implemented by a simple algorithm that randomly draws points from the full input space, then eliminates all points that do not meet the criteria of 5.3 and **??**. For our experiments the somewhat arbitrarily chosen g and p values were 25 and 0.25, respectively. The values were selected prior to running the algorithm on the test functions. The performance of the algorithm does not seem sensitive to the values.

5.1.2 Experiments

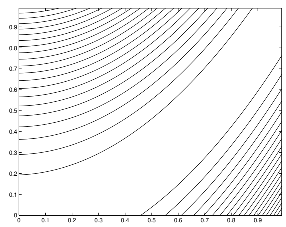
We compare PB1 with four versions of Nelder-Mead and three versions of PMAX. Nelder-Mead is the classic search algorithm described in Section 2.2.5. Amoeba2 is the same except it is made resistant to noise by doing two evaluations at each simplex vertex and taking their average. Amoeba4 and Amoeba8 similarly average four and eight evaluations at each vertex. All the Amoebas



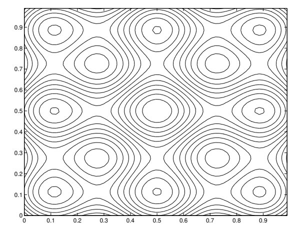
$$f_{gauss} = 1 - e^{(-20(x-0.25)^2+2(y-0.25)^2)}$$



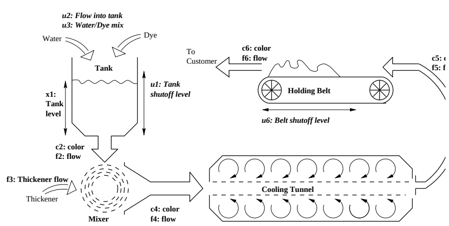
$$f_{discont} = \begin{cases} 1 - 2((x-0.5)^2 + (y-0.5)^2) & x < 0.5 \\ 0 & x \geq 0.5 \end{cases}$$



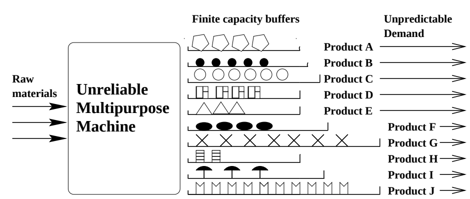
$$f_{Rosenbrock} = 10 - 100(y - x^2)^2 - (1-x)^2$$



$$f_{cosines} = 1 - (u^2 + v^2 - 0.3cos(3\pi u) - 0.3cos(3\pi v) + 0.7)$$
$$u = 1.6x - 0.5,\ v = 1.6y - 0.5$$



Chemical Plant Simulation (see text)



Multi-Buffer Simulation (see text)

Figure 5.2: The test functions used to evaluate performance. The task was to maximize the value of the test function.

begin with a medium-sized simplex started randomly in input space. The three different versions of PMAX used locally weighted quadratic regression with different kernel widths, meaning different positions on the bias-variance tradeoff.

All algorithms were tested on the functions from Figure 5.2; Table 5.1 contains the results. In all experiments we performed 25 independent runs of each optimizer, with each run consisting of 60 experiments. In addition to selecting the data points for the experiments, at every stage the optimizers also gave their estimate of the location of the optimum, and so we look at the true value of the underlying function at these estimates of the optimum. For PB1, the estimate of the optimum was taken to be the point in the reduced sample with the highest τ value. For the ith run of a particular optimizer, let s_i denote the mean of the true values at the estimates of the optimum. The figures in the columns are the mean s_i value of the optimizer on the final 15 of the 60 experiments over all 25 runs (i.e. $(\sum_i s_i)/25$).

PB1 performs well on all the functions, although it performs especially well in domains which are noisy, contain outliers, or contain discontinuities. An initial suspicion was that the symmetric nature of the minimum model that PB1 uses would render the algorithm susceptible to "ridges" in the objective function; we investigate this by adding two irrelevant dimensions to the f_{guass} function. This created a long ridge in four-dimensional space. PB1 fared as well as or better than all other methods except Amoeba on this function. The $f_{Rosenbrock}$ function contained a more pronounced and difficult ridge, which was responsible for PB1's worst performance, although PB1 became competitive in the noisy case.

Figure 5.2 shows a simulated version of a real industrial chemical process described in . The optimization task is to maximize the yield, which is determined by five parameters: reactant feed rate, mix-ratio, liquid level, thickener feed rate, and target inventory level. The yield is a very noisy and highly non-quadratic function; one input is almost irrelevant, the others are all important, and two of the inputs must run to their maximum legal value for best performance. The results are given in Table 5.1, and show that PB1 performs significantly better than all other methods tested.

Next, we examine a domain where experiments are time-consuming, a generalization of the multi-buffer machine task described in (this makes ten products instead of five). There are two inputs defining a simple parameterized policy for when to service an unreliable machine. This task is evaluated by a computationally expensive simulation; each function evaluation requires 10000 simulation steps. Evaluations are very stochastic (with highly non-Gaussian noise). The results are shown for runs of only 24 experiments. PB1 learns a

Table 5.1: The functions used are those from Figure 5.2, with the goal of maximization. Numbers in columns are mean score of last 15 experiments out of 60. The sample size is 25. 'Noise' is normally distributed with a standard deviation of 0.3. 'Outliers' indicates that with probability 0.1, noise will be an order of magnitude greater (standard deviation = 3.0). '2dims' indicates that two irrelevant input dimensions were added to the function. Significant ($p \leq 0.05$) results are denoted by an asterisk, and indicate that the algorithm tested significantly better than all other methods on that task. Due to the highly non-normal distribution of these means (bimodal in many cases) a nonparametric rank-sum test was used to determine significance.

	Amoeba 1	Amoeba 2	Amoeba 4	Amoeba 8	PMAX Global	PMAX Local	PMAX V.Local	Q2	PB1
f_{gauss}	**1.00**	0.99	0.80	0.61	0.99	0.99	**1.00**	**1.00**	**1.00**
f_{gauss}+Noise	0.36	0.52	0.52	0.45	0.42	0.57	0.54	0.69	**0.79***
f_{gauss}+Noise+Outliers	0.48	0.29	0.38	0.41	0.20	0.36	0.35	0.43	**0.72**
f_{gauss}+2dims	**0.99***	0.82	0.56	0.21	0.40	0.41	0.34	0.78	0.79
f_{gauss}+2dims+Noise	0.21	0.21	0.22	0.20	0.36	0.36	0.28	0.31	**0.60***
f_{gauss}+2dims+Noise+Outliers	0.17	0.14	0.13	0.16	0.20	0.25	0.19	0.18	**0.48***
$f_{discont}$	0.67	0.66	0.59	0.61	0.93	0.93	0.89	0.53	**1.00***
$f_{discont}$+Noise	0.76	0.81	0.78	0.76	0.86	**0.90**	0.78	0.86	**0.90**
$f_{discont}$+Noise+Outliers	0.71	0.73	0.70	0.74	0.65	0.71	0.65	0.71	**0.89***
$f_{Rosenbrock}$	**9.97***	9.75	8.94	5.40	8.96	9.23	9.43	9.71	9.59
$f_{Rosenbrock}$+Noise	9.60	9.58	8.93	5.42	8.89	9.28	8.93	**9.72***	9.46
$f_{Rosenbrock}$+Noise+Outliers	9.27	**9.56**	8.79	5.36	8.90	9.07	8.47	9.53	9.51
$f_{cosines}$	0.75	0.75	0.65	0.42	0.58	0.60	0.56	0.88	**0.90**
$f_{cosines}$+Noise	0.54	0.59	0.56	0.36	0.53	0.60	0.34	0.56	**0.79**
$f_{cosines}$+Noise+Outliers	0.41	0.39	0.38	0.34	0.33	0.34	0.21	0.29	**0.77***
Chemical Plant Simulation	26.39	30.04	25.24	17.90	40.33	39.52	24.80	42.79	**48.86***
Multi-Buffer Simulation	1.68	2.41	2.82	1.02	-1.38	-1.45	-1.52	3.06	**3.49**

good policy in these 24 experiments, i.e. a total of only 24×10000 simulation steps. This compares favorably with the tens of millions of simulation steps needed for reinforcement learning.

5.1.3 Summary

We have developed a statistical method for characterizing optima, pairwise bisection, which uses a Kendall's tau distribution. A new optimization algorithm based on this nonparametric approach uses this statistic for sample reduction and experiment selection, and it seems to be competitive with existing algorithms. Pairwise bisection attempts to introduce discrete reasoning into optimization by assuming a set of binary constraints on the minimum, namely, that the smaller member of every pair of data points is closer to a minimum. Nonparametric statistical reasoning decides whether a set of points indicates the presence of a minimum, and where to choose future experiments. The algorithm was compared to Nelder-Mead simplex search, PMAX, and Q2, and its performance was in general very good: it performed very well in the presence of a discontinuity, and was much more robust to noise and outliers than the other algorithms. PABI's worst performance was on the Rosenbrock function, probably because PABI assumes a symmetric space. This can be avoided by projecting the control variables into a suitable feature space.

Chapter 6

Conclusions and Future Work

We explored two new optimization techniques, GMorph and PABI. The first is a application to the domain of determining galactic morphology from images using nearest neighbor, and the second is a nonparametric optimization method.

GMorph allows one to search in a much reduced search space over those searched by current methods, and so is much more efficient in terms of function evaluations. The accuracy of GMorph is reported, as are the systematic errors that arise in different noise conditions and with different amounts of blurring. These tables can be used to make decisions regarding the amount of signal-to-noise required for a given level of accuracy.

PABI has been introduced and described as a nonparametric optimization routine. It uses a 'model-free' statistic, Kendall's tau, to compute likelihoods for different optima. The performance of PABI in several simulated environments is superior to Amoeba, PMAX, and Q2, and it has been shown to be particularly resistant to the presence of noise and outliers.

6.1 Future Research

Future research for GMorph includes techniques for speeding up the nearest neighbor search. A kd-tree could give a rough estimate of the nearest neighbors, although the PSF and linear regression aspects of the task would thwart a traditional implementation. A local descent step at the end of each nearest neighbor search would give a more accurate estimate of the maximum likelihood parameters, however, it may not offset the cost in speed. An easy gain in efficiency can be had by translating the code from matlab to C.

An interesting idea would be to use the distribution of eigencoefficients to identify anomalies. Since the eigencoordinates of real galaxies are clustered and have structure even within the subspace Φ, one could use any observed deviations from this as an indicator of an unclassified event. One would first use synthetic and/or actual galaxies to create a prior on eigencoefficients, then during run-time use this to determine the likelihood of the eigencoordinates of every new galaxy seen. It would also need to be determined whether eigengalaxies have an advantage over, for instance, the Fourier coefficients of residual images.

Also, since the errors in galaxy images are generally Poisson distributed, the variance is proportional to the intensity of the signal. This means noise is heteroskedastic across the image. We could try taking the log of all the values, rendering the errors log-normal and hopefully of uniform variance. Another approach is a weighted least squares regression to the data, with the weights being inversely proportional to the estimated standard deviation of each pixel.

Future research for PABI would include further exploration into the domain of optimization in similarity-based spaces. Adaptively changing the similarity metric as the search gets more local would be interesting as well [18]. Allowing the ROI search to be more flexible, e.g., moving, instead of just subtracting points would allow a more comprehensive search. However, all of this increases the risk of over-fitting, so new constraints on the search should be developed at the same time. The further generalization of the PABI and Q2-like set of optimization algorithms would be a valuable addition to the field. Broadly speaking, the more interesting direction of research, to us, is towards optimization algorithms inspired by the fields of Artificial Intelligence and statistics. Optimizing human experience is a common goal, and similarity-based search spaces, nonlinearities, noise, and outliers are widespread in the sphere of human experience.

Bibliography

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6:37–66, 1991.

[2] Brigham Anderson, Andrew Moore, and David Cohn. A Nonparametric Approach to Noisy and Costly Optimization. In *International Conference on Machine Learning*, 2000.

[3] C. G. Atkeson, A. W. Moore, and S. A. Schaal. Locally Weighted Learning. *AI Review*, 11:11–73, April 1997.

[4] C. G. Atkeson, A. W. Moore, and S. A. Schaal. Locally Weighted Learning for Control. *AI Review*, 11:75–113, April 1997.

[5] G. E. P. Box and N. R. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, 1987.

[6] George E. P. Box and Norman R. Draper. *Evolutionary Operation: A Statistical Method for Process Improvement*. John Wiley & Sons, 1969.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, September 1997.

[8] C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):129–156, 1999.

[9] David A. Cohn. Minimizing Statistical Bias with Queries. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.

[10] W. Daelemans, J. Zavrel, and S. Berck. Mbt: A memorybased part of speech tagger-generator, 1996.

[11] G. de Vaucouleurs. *Monthly Notices of the Royal Astronomical Society*, 1953.

[12] R. O. Duda and P. E. Hart. *Pattern Classification, Second Edition*. John Wiley & Sons, 2001.

[13] K. C. Freeman. On the Disks of Spiral and s0 Galaxies. *Astrophysical Journal*, 160:811, 1970.

[14] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

[15] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

[16] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*. Assn for Computing Machinery, April 1984.

[17] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[18] Nicholas Howe and Claire Cardie. Examining locally varying weights for nearest neighbor algorithms. In *ICCBR*, pages 455–466, 1997.

[19] E. Hubble. *The realm of the nebulae*. Yale University Press, 1936.

[20] Terry Jones and Stephanie Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Kaufmann, 1995.

[21] L. P. Kaelbling. Learning in Embedded Systems. . PhD. Thesis; Technical Report No. TR-90-04, Stanford University, Department of Computer Science, June 1990.

[22] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30:81–93, 1938.

[23] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *Annals of Mathematical Statistics*, 23:462–466, 1952.

[24] S. Kirshner, I. Cadez, P. Smyth, and C. Kamath. Learning to classify galaxy shapes using the EM algorithm. In *Advances in Neural Information Processing Systems*, volume 15. Morgan Kaufmann, 2002.

[25] J. Kormendy. Brightness distributions in compact and normal galaxies. III - Decomposition of observed profiles into spheroid and disk components. *Astrophysical Journal*, 217:406–419, 1977.

[26] H. Kushner and D. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.

[27] M.M. Loeve. *Probability Theory*. Van Nostrand, Princeton, 1955.

[28] L.B. Lucy. An iterative technique for the rectification of observed distributions. *Astronomical Journal*, 79:745, 1974.

[29] D.W. Marquardt. *Journal of the society for industrial and applied mathematics*, 11:431–441, 1963.

[30] N. Metropolis, A.W. Rosenbluth, M. N. Rosenbluth, A.H. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1958.

[31] Melanie Mitchell, John H. Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing? In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 51–58. Morgan Kaufmann, 1994.

[32] Baback Moghaddam and Alex Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):696–710, 1997.

[33] A. W. Moore. Efficient Memory-based Learning for Robot Control. PhD. Thesis; Technical Report No. 209, Computer Laboratory, University of Cambridge, October 1990.

[34] A. W. Moore. Very fast mixture-model-based clustering using multiresolution kd-trees. In M. Kearns and D. Cohn, editors, *Advances in Neural Information Processing Systems 10*, pages 543–549, San Francisco, April 1999. Morgan Kaufmann.

[35] A. W. Moore, D. J. Hill, and M. P. Johnson. An Empirical Investigation of Brute Force to choose Features, Smoothers and Function Approximators. In S. Hanson, S. Judd, and T. Petsche, editors, *Computational Learning Theory and Natural Learning Systems, Volume 3*, pages 361–379. MIT Press, 1994.

[36] A. W. Moore and J. Schneider. Memory-based Stochastic Optimization. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Neural Information Processing Systems 8*, pages 1066–1072. MIT Press, 1996.

[37] A. W. Moore, J. Schneider, J. Boyan, and M. S. Lee. Q2: A memory-based active learning algorithm for Blackbox Noisy Optimization. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 386–394. Morgan Kaufmann, 1998.

[38] S. Nayar, H. Murase, and S. Nene. Parametric appearance representation, 1996.

[39] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.

[40] C. Y. Peng, L. C. Ho, C. D. Impey, and H. Rix. Detailed Structural Decomposition of Galaxy Images. *Astronomical Journal*, 124:266–293, 2002.

[41] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, WA, June 1994.

[42] Cary D. Perttunen and Bruce E. Stuckman. The Rank Transformation Applied to a Multiunivariate Method of Global Optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:1216–1220, 1990.

[43] Roger G. Peterson. *Design and Analysis of Experiments.* Marcel Dekker, 1985.

[44] M.J.D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 17:155–162, 1964.

[45] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[46] Rajesh P. N. Rao and Dana H. Ballard. Natural basis functions and topographic memory for face recognition. In *IJCAI*, pages 10–19, 1995.

[47] K. Ratnatunga, R. Griffiths, and E. Ostrander. Disk And Bulge Morphology Of Wfpc2 Galaxies: The Hst Medium Deep Survey Database. *accepted to Astronomical Journal*, 1999.

[48] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[49] Dan Roth. Memory based learning in NLP. Technical Report 2125, Urbana, Illinois, 1999.

[50] M. Salganicoff and L. H. Ungar. Active Exploration and Learning in Real-Valued Spaces using Multi-Armed Bandit Allocation Indices. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 480–488. Morgan Kaufmann, 1995.

[51] T. Sim, R. Sukthankar, M. Mullin, and S. Baluja. High-performance memory-based face recognition for visitor identification, 1999.

[52] Luc Simard, Christopher N. A. Willmer, Nicole P. Vogt, Vicki L. Sarajedini, Andrew C. Phillips, Benjamin J. Weiner, David C. Koo, Myungshin Im, Garth D. Illingworth, and S. M. Faber. The Deep Groth Strip Survey II. Hubble Space Telescope Structural Parameters of Galaxies in the Groth Strip. *Astrophysical Journal Supplements*, 142(1), 2002.

[53] C. Stanfill and D. Waltz. Towards Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1228, December 1986.

[54] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, 1995.

[55] A.N. Tikhonov and V.Y. Arsenin. *Solutions of ill-posed problems.* Winston, Washington DC, 1977.

[56] Aimo Torn and Antanas Zilinskas. *Global Optimization: Lecture Notes in Computer Science, vol. 350.* Springer-Verlag, 1987.

[57] D. Vanderbilt and S.G. Louie. A monte carlo simulated annealing approach to optimization over continuous variables. *Journal of Computational Physics*, 56:259–271, 1984.